University of Edinburgh

Department of Computer Science

Proof Techniques for CCS

by

Michael Thomas Sanderson

Thesis

CST-19-82

James Clerk Maxwell Building,
The King's Buildings,
Mayfield Road,
Edinburgh,
EH9 3JZ.

November, 1982

PROOF TECHNIQUES FOR CCS

Michael Thomas Sanderson

Ph. D.
University of Edinburgh
1982

(Graduation date July 1983)

ABSTRACT

Proofs of observational equivalence of behaviour expressions in Milner's Calculus of Communicating Systems can be quite lengthy, and as larger and more practical systems of agents are considered the need for shorter proof techniques becomes more important. In this thesis a number of results about the calculus are proved which give rise to give more natural techniques. Three principal areas of research are presented:

(i) A study of strong confluence and determinacy is made, extending Milner's work to the whole calculus - the appropriate modifications to take value-passing into account are motivated and defined, and a strong confluence theorem is proved. It is shown that a useful subcalculus of CCS is strongly confluent.

(ii) An investigation into criteria for uniqueness of solution of equations of the form b = F[b] is performed. To do this a concept of derivations of an agent A "causing" derivations of F[A] is defined; using this, conditions are imposed on F which imply uniqueness, and a study follows of how these conditions relate to the structure of F.

(iii) By using an alternative, stronger, definition of observational equivalence as a maximal fixed point it is found that equivalences can be demonstrated by constructing bisimulations between agents, and results leading to an algorithm for such constructions are presented. Also, using this alternative definition a weaker form of confluence can be defined very easily, and this is investigated.

The theoretical material in this thesis is supplemented by examples demonstrating how the results proved can be applied to give proof techniques for use within the calculus.

DECLARATION

The research presented in this thesis is all my own, and previously unpublished, with minor exceptions as indicated in the text.

CONTENTS

1 INTRODUCTION

The study of concurrency in computing systems is an active area of current research. From the theoretical viewpoint a number of different treatments have been developed in recent years, among them the net thoery of Petri and his colleagues at Bonn [Pet], Hoare's Communicating Sequential Processes [Hoa], the path expressions developed by Campbell and Habermann at Newcastle [C+H] and Robin Milner's Calculus of Communicating Systems [Mil4]. This thesis presents further developments to the latter approach, investigating several properties of the calculus which lead to useful proof techniques.

Milner's calculus, generally referred to as CCS, is an algebraic treatment of concurrency: agents are constructed from behaviour identifiers and a set of operators. An operational semantics is defined in terms of derivations, or observable actions, of agents, these capabilities being defined structurally by a set of rules for deducing the possible derivations of a system from the derivations of its component parts. Using these derivations a concept of observational equivalence of behaviours is defined - two systems are equivalent if they are indistinguishable by their actions as viewed by an external observer. Concurrency is obtained by a composition operator which combines two agents allowing them to communicate with each other as well as with their environment; such internal communications of a system are not observable externally.

CCS was developed from earlier work by Milner and George Milne [M+M] in which a model based on a powerdomain construction was presented for concurrent computing agents, this work itself being an adaptation of earlier work by Milner [Mil1]. With notational changes for syntactic clarity this early research led to a language for behaviour expressions [Mil2] which was removed from the powerdomain model. In order to consider the correctness of

various possible interpretations of this language the notion of observational equivalence was introduced and from this grew the calculus CCS. Algebraic laws were discovered and a set of these was shown to be complete for a simplified class of finite behaviours [H+M].

Milner in [Mil4] presents several examples of behaviours together with algebraic proofs that they satisfy certain specifications. These proofs in the calculus frequently involve lengthy manipulation and as larger and more practical systems of agents come under consideration it seems likely that the amount of work involved in proving equivalences will increase substantially, unless techniques are developed to reduce it. Hence there is a need for a framework of results about the calculus which can lead to the development of new, and shorter, proof techniques. Research to this end was commenced by Milner in [Mil4] with the introduction of the concepts of confluence and determinacy in the final chapter, where it was shown that strongly confluent behaviours satisfy a theorem which facilitates the shortening of proofs of observational equivalence. The study was limited to a restricted subcalculus of CCS; in this thesis the definitions and results are extended to the whole calculus, resulting in an analogous strong confluence theorem, and it is shown that a useful class of behaviour expressions in the calculus has the property of strong confluence.

Two other principal areas of research are presented in this thesis; the first of these is an investigation of criteria for the uniqueness of solution of recursive behaviour-defining equations of the form $b = F[b]$; when the solutions of such an equation are unique up to observational equivalence then F is said to be a contracting transformation. The proof that certain very simple equations have unique solutions was set as an exercise in [Mil4]; here it is shown that a much larger class of transformations has the contracting property. The significance of this is that if two behaviour expressions can be shown to satisfy an equation $b = F[b]$

where F is a member of this class then it can be concluded that the behaviours are observationally equivalent.

The third major area of research reported in this thesis concerns the implications of an alternative definition of observational equivalence. The original definition uses the intersection of a decreasing sequence of equivalences each obtained from the previous one by a defining relation. David Park suggested that defining observational equivalence to be the maximal fixed point of this relation under the partial ordering of set inclusion would give rise to favourable properties. The new equivalence is stronger than the original one and hence in order to prove that two agents are equivalent in the original sense it is sufficient to show that they satisfy the fixed-point equivalence. By standard fixed-point theory this can be done by proving that they satisfy any equivalence which is a fixed point of the defining relation, such an equivalence being referred to as a bisimulation [Par]. The research here concerns techniques for constructing bisimulations; an algorithm is presented which should not be too difficult to mechanise. As an example of its use a bisimulation is constructed between two recursively-defined behaviour expressions; although this involves lengthy manipulation no other technique is known for proving their observational equivalence.

The definition of observational equivalence as a maximal fixed point has been used by Milner in a recent publication [Mil5] and it seems likely that this definition will supersede the original one in future research. (For most applications the choice of equivalence is immaterial since restricted to finite behaviours the two equivalences are the same, as they are for all practically-inspired behaviour expressions so far encountered; the simplest example known of a pair of agents which are equivalent under the original definition but not under the new one is extremely complex and very artificial in its appearance.) However, most of the research on confluence and determinacy and on criteria

for contracting transformations for this thesis was performed before the emergence of the new approach; hence it is presented in terms of the original equivalence. Nevertheless, the fixed-point definition does seem very appropriate to a treatment of observation confluence, a weaker form of confluence than the strong confluence previously referred to, and the implications of this are presented here.

Throughout this thesis examples are given to complement the theoretical study and demonstrate how the results proved give rise to useful proof techniques. Many of the agents considered in these examples are taken from examples in [Mil4] so that a contrast emerges between the use of these techniques and the manipulation involved in proving observational equivalences directly from the algebraic laws of the calculus.

The remainder of this introduction is devoted to an outline of the contents of each chapter of the thesis.

SYNTAX AND PROPERTIES OF CCS

Although familiarity with Milner's Calculus of Communicating Systems is required for a full appreciation of this research, a summary of definitions and the main algebraic laws and other results that are required in proofs and examples in this thesis is presented in Chapter 2, along with references to the relevant sections of [Mil4] from which further details can be obtained. After a brief informal introduction to the syntax and intuitive interpretation of behaviour expressions formal definitions of underlying sets and essential auxiliary notations are given, followed by an introduction to the operators used in building expressions: prefix operators representing action and communication, + representing ambiguity or choice of action, | representing concurrent composition, \ representing restriction or the limiting of certain communication capabilities to internal

communication within an agent, preventing external observation, and relabelling operators useful particularly in combining copies of identical agents to provide the appropriate "linking". Other syntactic devices such as the use of behaviour identifiers and conditional clauses are also introduced. A summary of the derivation rules for determining the behaviour of a system of agents from the behaviours of its individual components is given, the behaviour of an agent being regarded as its ability or inability to undergo certain actions or communications with an external observer. The definitions of observational equivalence, denoted by $\approx$, and a stronger equivalence, $\sim$, known as strong congruence, are presented and it is shown how observational equivalence is extended to a congruence relation. Finally, the algebraic laws of these equivalences together with other properties that are needed in the research that follows are listed together with page references to [Mil4] indicating where their proofs can be found.

Throughout the remainder of this introductory chapter it is assumed that the reader is familiar with the notation of CCS.

CONFLUENCE AND DETERMINACY

The third chapter of this thesis is devoted to the extension of the concepts of strong confluence and determinacy, introduced for pure CCS (a subcalculus limiting communication to pure synchronisation and excluding variables and value-passing) by Milner in [Mil4], to the full calculus, and proving results similar to those presented by Milner to enable the application of confluence properties to proof techniques. The concepts encapsulate properties associated with an intuitive notion of behaviours being deterministic, and the essential property of a strongly confluent agent, A, under Milner's definition, is that whenever $A \xrightarrow{\mu} B$ and $A \xrightarrow{\nu} C$ then either $\mu = \nu$ or there exist agents D and E such that $B \xrightarrow{\nu} D$, $C \xrightarrow{\mu} E$ and D and E are strongly congruent.

This simple definition is soon seen to be inappropriate for the full calculus, the condition $\mu=\nu$ being too restrictive when in cases where the guards μ and ν are not equal but involve the same label. Consider for example the simple buffer defined by

$$A \Leftarrow \alpha x.\bar{\beta} x.NIL$$

(assuming that the set of values associated with the α and $\bar{\beta}$ labels is a subset of the integers); then $A \xrightarrow{\alpha 5} \bar{\beta} 5.NIL$ and $A \xrightarrow{\alpha 7} \bar{\beta} 7.NIL$, and as $\bar{\beta} 5.NIL$ can undergo no $\alpha 7$-action A does not satisfy the definition. For confluence to be of any practical use in equivalence-proving it is clearly desirable that an agent such as this should be admitted by the definition; hence adaptations of the conditions are needed. These should not allow a behaviour such as

$$B \Leftarrow \bar{\beta} 5.NIL + \bar{\beta} 7.NIL$$

to be confluent, since this does not satisfy any intuitive concepts of being deterministic, so care must be taken in the treatment of positive and negative labels (that is of the form α and $\bar{\alpha}$ respectively).

After the definition chosen has been presented and justified, results are proved leading to a strong confluence theorem analogous to that presented by Milner for the pure calculus. This theorem states that if a behaviour A is strongly confluent and $A \xrightarrow{\tau} B$ for some behaviour B then A and B are observationally equivalent. Investigations are then made into which operators in the calculus preserve strong confluence, in order to obtain a class of confluent behaviours to which the theorem can be freely applied. As with the pure calculus it is found useful to define a related concept of strong determinacy; again adaptations have to be made to Milner's definition. It is found that the ambiguity operator (+) does not preserve strong confluence, even in the presence of determinacy, so in order to admit behaviours with some

degree of ambiguity to the class of confluent agents it is necessary to introduce a derived operator, known as composite guarding. The definition given by Milner for the pure calculus proves inadequate because of problems involved with the binding of free variables; hence a number of conditions have to be imposed on composite guards, and these are presented and justified by examples of violations occurring in their absence. As in [Mil4] it is proved that all the operations of the derived calculus do preserve strong confluence and determinacy, hence all behaviours in the derived calculus are confluent, enabling the application of the strong confluence theorem whenever it is of use in proofs of observational equivalence. As an example of its application an example is presented: a buffer is constructed and it is shown that it meets a formal specification.

UNIQUENESS OF SOLUTION OF BEHAVIOUR EQUATIONS

In chapter 4 a study is made of criteria necessary to ensure that an equation of the form $b \simeq F[b]$ has a unique solution up to observational equivalence. It is not difficult to see that equations such as $b \simeq \alpha.b$ and $b \simeq \alpha.(\beta.b + \gamma.NIL)$ do have unique solutions, whereas $b \simeq \tau.b$ does not; it would therefore seem reasonable to anticipate that the equation $b \simeq F[b]$ has a unique solution provided the behaviour b is guarded by a label other than τ in the expression denoting $F[b]$. This condition is probably sufficient in the absence of the composition operator, but it will be demonstrated that the equation

$$b \simeq \alpha.(b|\bar{\alpha}.NIL)\backslash\alpha$$

has many different solutions which are not observationally equivalent; in fact it is satisfied by any agent $\alpha.A$ such that α and $\bar{\alpha}$ are not in the sort of A.

In order to investigate conditions on a context $F[\ ]$ sufficient to ensure that all solutions of $b \approx F[b]$ are observationally equivalent (when F is said to be contracting) a study is made of how actions of a behaviour $F[B]$ may be caused by the actions of B. This research is limited to the pure calculus, and to simple contexts F where the symbol b occurs only once in the expression represented by $F[b]$. A "causing" property written $F[\ ] \xrightarrow[\mu']{\nu} F'[\ ]$ is defined, having the meaning that whenever $x \xrightarrow{\mu'} y$ then $F[x] \xrightarrow{\nu} F'[y]$; for example if $F[b] \equiv b[\beta/\alpha]$ then $F[\ ] \xrightarrow[\alpha]{\beta} F[\ ]$. After the properties of this definition are investigated it is extended to $F[\ ] \overset{t}{\underset{s}{\Rightarrow}} F'[\ ]$ for non-unit-length derivations. The obvious definition analogous to that of $F[\ ] \xrightarrow[\mu']{\nu} F'[\ ]$ is unsuitable as the required properties cannot be proved due to the invisibility of τ-actions in derivations $x \xrightarrow{s} y$; hence a definition making use of $F[\ ] \xrightarrow[\mu']{\nu} F'[\ ]$ is chosen. Great care has to be taken to ensure that this does produce a "causing" property, but after the appropriate definition has been justified informally it is found that it does facilitate the proof of the desired properties.

Using the properties of the concepts of causing of derivations a theorem is proved that under two conditions on the context F it can be concluded from $A_1 \approx F[A_1]$ and $A_2 \approx F[A_2]$ that $A_1 \approx A_2$. The two conditions are expressed in order to facilitate the proof and do not relate directly to the structure of F, so the remainder of the chapter is devoted to an examination of their implications. The theorem is proved using induction on k to show that $A_1 \approx_k A_2$ for all k, and to do this requires a further induction on derivation strings encountered. The induction is performed on a measure #, and the first condition imposed on F is that whenever $F[\ ] \overset{t}{\underset{s}{\Rightarrow}} F'[\ ]$ then either F' is constant or $\#s < \#t$. Taking # to be the length of the string it is found that the set of contexts satisfying this condition is too small; hence an alternative measure is needed. By considering examples it is discovered that defining #s to be the number of occurrences of a fixed label λ_0 in s appears to be suitable. Restrictions on the structure of F which are needed in order to satisfy the condition are discovered

by investigating examples, and it is shown that the restrictions chosen are sufficient. Additional constraints on the structure of F are needed to ensure that the second condition for the main theorem is satisfied; it is found that these are easier to find, and combining the two sets of restrictions a class of contexts is found for which the theorem applies. A simple example of its use is presented, followed by a discussion of the further research necessary in order to extend the concepts introduced to the full calculus and to more general transformations than the simple contexts considered.

AN ALTERNATIVE DEFINITION OF EQUIVALENCE

The observational equivalence relation $\simeq$ used by Milner in [Mil4] is defined as the intersection of a decreasing sequence of equivalences $\simeq_k$, where $\simeq_0$ is the universal relation and each equivalence is obtained from its predecessor by a defining relation of the form

$$\simeq_{k+1} = E'(\simeq_k).$$

David Park suggested that it would be useful to consider another equivalence, the maximal fixed point of the equation

$$R = E'(R)$$

using the partial ordering of set inclusion. This equivalence is denoted by $\simeq_F$, and in chapter 5 we show that it is stronger than $\simeq$, an example being presented to show that the two equivalences are not equal. In order to prove the observational equivalence of two agents A and B it is sufficient to demonstrate that $A \simeq_F B$, and an investigation of the properties of $\simeq_F$ shows that this can often be easier than attempting a direct algebraic proof of the equivalence.

Two properties of the fixed-point equivalence are particularly important: the first is a standard result from fixed-point theory that if $R \subseteq E'(R)$ then $R \subseteq \approx_F$, and the second is that a simpler defining relation E', involving only derivation strings of length 0 or 1, can be shown to have the property that $R \subseteq E(R)$ if and only if $R \subseteq E'(R)$. As a conclusion it follows that to prove $A \approx_F B$ it is sufficient to demonstrate the existence of a relation R such that $R \subseteq E(R)$ and $\langle A,B \rangle \in R$. Following Park in [Par] such a relation is referred to as a bisimulation and the main aim of the research presented in chapter 5 is to present techniques for the construction of bisimulations. Two simple examples are considered: one a practical one of proving that a simple scheduler satisfies a specification, and the other a more theoretical one of constructing one bisimulation from another in order to prove that the guarding operator preserves $\approx_F$. In order to develop formal techniques results are proved which allow the checking of whether a relation is a bisimulation to be performed by considering only derivations of the form $A \xrightarrow{\lambda} C$ and $A \xrightarrow{\tau} C$ rather than $A \overset{\mu}{\Rightarrow} C$, under certain reasonable assumptions about A. These enable the construction of an algorithm for the step-by-step construction of bisimulations starting with the relation $\{\langle A,B \rangle\}$ and adding further ordered pairs as necessary. This algorithm is expressed formally for the pure subcalculus, and the necessary adaptations to extend it to the full calculus are presented by means of an example of its use. This example involves lengthy manipulation but does succeed in proving an equivalence between two behaviours for which no other technique has been succesful; furthermore it is believed that the algorithm should be mechanisable without too much difficulty, so the lengthy manipulation is not really a problem.

In chapter 6 other implications of the alternative approach to observational equivalence are considered. It is found that by being able to work with derivation strings of length 0 and 1 only it is possible to simplify considerably the definition of observation confluence introduced by Milner in [Mil4]. This form

of confluence is weaker than the strong confluence studied in chapter 2, but it still leads to an observation confluence theorem similar to the strong confluence theorem discussed above. Working in a maximal-fixed-point environment it is appropriate to define observation confluence in terms of a fixed point rather than by defining a sequence of k-confluence properties; this is done initially for the pure calculus and then extended to the full CCS, the set of observation-confluent agents being defined as the maximal fixed point of a relation $\dot{\Phi}$. In order to prove a behaviour expression confluent it is then sufficient to show that it is a member of a set S such that $S \subseteq \dot{\Phi}(S)$; the construction of such sets is generally quite straightforward. As an example the behaviour of a queue is specified and the observation confluence theorem is used in proving that a constructed agent satisfies the specification. The construction cannot be expressed in terms of the derived confluent subcalculus of chapter 3, and although it is believed that it is strongly confluent it is unnecessary to demonstrate this, which would involve more work than the construction of the set $S \subseteq \dot{\Phi}(S)$ necessary to prove observation confluence. Hence the fixed-point definition of observation confluence is seen to be of more general use than the the inductively-defined strong confluence; however, strong confluence is of value when working with agents constructed in the derived subcalculus.

2 THE SYNTAX AND SEMANTICS OF CCS

INTRODUCTION

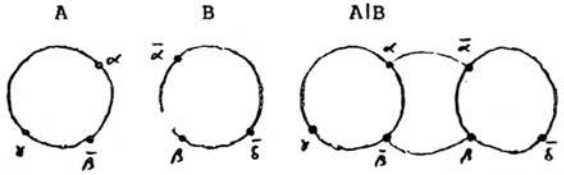
In this chapter the syntax and operational semantics of CCS will be introduced formally, and rather briefly, together with a number of properties and results that are required in this thesis. For greater detail, and proofs (which are omitted here), the reader is referred to [Mil4], in particular chapters 5 and 7.

The terms of CCS are behaviour expressions, on which a schema of derivation rules is imposed, enabling the definition of observational equivalence of behaviours and hence presenting an operational semantics. Most of the properties that will be quoted are algebraic laws of observational equivalence and associated stronger equivalences and congruences.

The terms of the calculus are built up from a number of operators, of arity 0, 1 and 2, and also variable symbols. The operators can be split into two groups, dynamic and static. The static operators include $\backslash\alpha$ (where α ranges over a set Δ which will be defined subsequently), of arity 1 and written postfix, and $|$, an infix operator of arity 2. From their dynamic structure behaviours have certain communication or synchronisation capabilities, and informally they may be regarded as having communication ports, labelled by elements of the form α or $\bar{\alpha}$, where $\alpha \in \Delta$. It is then possible to represent behaviours pictorially, showing these communication ports as in [Mil3].

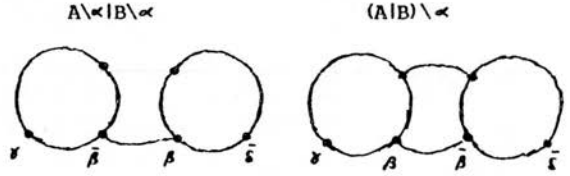
The $|$ operator represents the composition of two behaviours, enabling communication to take place between them, and allowing their dynamic actions to interleave freely. Communication can occur between two ports α and $\bar{\alpha}$ (for $\alpha \in \Delta$) - the α port "offers" a value and the $\bar{\alpha}$ port "accepts" one. If a behaviour A has sort $\{\alpha, \bar{\beta}, \gamma\}$ (that is its communication capabilities are via α, $\bar{\beta}$,

and γ ports) and B has sort $\{\bar{\alpha}, \beta, \bar{\delta}\}$ then A, B and A|B may be represented by the following diagrams:



In order that diagrams such as these are not ambiguous it is necessary that the | operator is both commutative and associative; that is A|B and B|A are in some sense equivalent, and likewise A|(B|C) and (A|B)|C. This will be found to be the case when observational equivalence is defined.

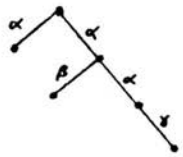
The $\backslash\alpha$ operator prevents a behaviour from communicating externally via its α and $\bar{\alpha}$ ports and can hence be used to ensure that certain communications of behaviours composed by the | operator occur internally. With A and B as defined above the behaviours $A\backslash\alpha|B\backslash\alpha$ and $(A|B)\backslash\alpha$ may be represented diagrammatically:



In defining the behaviours above it has been assumed that \ binds more tightly than |; the precedences of operators will be given later.

The main dynamic behaviour operation is known as "guarding", the simplest form of which is $\alpha.A$, which will communicate via its α port and then behave like A. This is a case of pure synchronisation; values may be associated with the communication

capabilities giving guards such as αx and $\bar{\alpha}7$. Another important dynamic behaviour operation is ambiguity, of arity 2 and written as an infix +; the behaviour A + B may behave as either A or B. There is also a dynamic behaviour operator NIL of arity 0, which can undergo no communications. Behaviours constructed from these dynamic operators may be represented as trees in the obvious manner, for example $\alpha.\mathrm{NIL} + \alpha.(\beta.\mathrm{NIL} + \alpha.\gamma.\mathrm{NIL})$ would be illustrated as:



In examples, most behaviours under consideration will be constructed by applying the static operators to behaviours built from the dynamic operators.

LABELS AND SORTS

Before presenting the syntax of CCS formally it is necessary to introduce some preliminary concepts and define certain sets that will be used. It is assumed that there is a finite set, Δ, of names, members of which are generally represented by $\alpha, \beta, \gamma, \ldots$ Additionally a set $\bar{\Delta}$ is assumed, in bijection with and disjoint from Δ. This is referred to as the set of co-names and its elements are represented by $\bar{\alpha}, \bar{\beta}, \bar{\gamma}, \ldots$, with $\bar{\ }$ denoting the bijection. The inverse bijection is also denoted by $\bar{\ }$, hence, for example, $\bar{\bar{\alpha}} = \alpha$.

Labels are defined to be elements of the set $\Lambda \cup \{\tau\}$, where $\Lambda = \Delta \cup \bar{\Delta}$ and $\tau \notin \Lambda$. A label is said to be positive if it is a member of Δ and negative if it is in $\bar{\Delta}$; α and $\bar{\alpha}$ are referred to as complementary labels - α is the complement of $\bar{\alpha}$ and vice versa.

Positive labels may be used to bind variables; if α is such a label then its complement qualifies value expressions of the same type. Informally this represents the acceptance and offering of values, or "input" and "output". (The full calculus as defined by Milner in [Mil4] also allows tuples of variables and expressions to accompany appropriate labels; for convenience and to simplify notation this will be ignored in this thesis.) In some cases labels are used without values as pure synchronisation capabilities; however when convenient it is possible to regard these as being accompanied by a value chosen from a singleton set (or a variable whose value can range over such a set). The τ label (informally) represents internal or unobservable action; it is never accompanied by a variable or value expression. A label together with a variable or value-expression as appropriate is known as a guard. μ is used to range over guards; if a guard μ contains the label λ then name(μ) is defined to be equal to λ in the case of a positive label, but equal to $\bar{\lambda}$ if λ is negative (so name(μ) is always positive).

A sort L is a subset of Λ; each behaviour expression B will be assigned a sort L(B), although it is convenient to allow B to possess all larger sorts as well (using B:L to denote "B possesses sort L" this means B:M whenever $L(B) \subseteq M$). A relabelling $S:L \to M$ from sorts L to M is a bijection which preserves complements (that is $S\bar{\lambda} = \overline{S\lambda}$ for all $\lambda \in L$) and respects positive and negative properties of labels that are used for value-communication (and also respects the types of the variables and expressions that can accompany such labels). A relabelling S may be written in the form $\beta_1 \ldots \beta_n / \alpha_1 \ldots \alpha_n$, which means that $S\alpha_i = \beta_i$ for $i \in \{1,\ldots,n\}$ and $S\alpha = \alpha$ for $\alpha \notin \{\alpha_1,\ldots,\alpha_n\}$.

BEHAVIOUR EXPRESSIONS

Behaviour expressions in CCS are formed using behaviour operators, parameterised behaviour identifiers and conditionals.

It is assumed that there is a collection of behaviour identifiers b each having a preassigned arity n(b) and sort L(b), and that the meaning of each of these identifiers is given, possibly recursively, by a behaviour expression. There are certain restrictions on expressions accepted as defining the meaning of a behaviour identifier; these will be introduced later.

The six forms of behaviour operator are NIL, of arity 0 (representing inaction), + (representing ambiguity or choice) and | (representing composition) of arity 2, both written as infix operators, and three families of operators of arity 1: action (or guarding), written as $\alpha x.B$ or $\bar{\alpha} E.B$ (where x and E are a variable and an expression respectively, of the type appropriate to the label α) or $\tau.B$; restriction, written as $B\backslash\alpha$ (and preventing external communication via α and $\bar{\alpha}$ labels); and relabelling, written as B[S] where S is a relabelling as defined above (which renames the labels in the behaviour according to S). A conditional behaviour expression is written in the form

if E then B else B'

where E is a boolean expression (and has the expected meaning, that is the behaviour behaves as either B or B', depending on the value of E). The precise meaning of each behaviour operator will be given in the next section, when the operational semantics of behaviours is introduced via derivation rules.

The sort of a behaviour expression is defined by structural induction, the sorts of the individual behaviour constructions being given by the following table, where L(B) is used to represent the sort of the behaviour B:

Expression	Sort
NIL	ϕ
B + B'	$L(B) \cup L(B')$
$\alpha x.B$	$L(B) \cup \{\alpha\}$

$\tilde{\alpha}E.B$	$L(B) \cup \{\tilde{\alpha}\}$
$\tau.B$	$L(B)$
$B\|B'$	$L(B) \cup L(B')$
$B\backslash\alpha$	$L(B) - \{\alpha,\tilde{\alpha}\}$
$B[S]$	$\{S\lambda : \lambda \in L(B)\}$
if E then B else B'	$L(B) \cup L(B')$
$b(E_1,\ldots,E_{n(b)})$	$L(b)$

A behaviour B is also allowed to possess any larger sort that L(B); writing B:L to mean "B has sort L", then B:M whenever $L(B) \subseteq M$.

The normal order of precedence of behaviour operators is given by the following list; the first-listed operators bind most tightly:

restriction or relabelling
action
composition
summation
conditional

The normal precedences can of course be over-ridden by the use of parentheses.

It was mentioned earlier that only certain behaviour expressions are admissible as the defining-clauses of behaviour identifiers. The ones that are not accepted are those in which a behaviour calls itself recursively without passing a guard, for example

$$b(x) \Leftarrow \alpha x.NIL + b(x+1)$$

is not allowed, nor the mutually recursive pair

$b_1 \Leftarrow b_2 + \alpha.b_3$
$b_2 \Leftarrow b_1 | \beta.b_4$.

The reasons for this are explained by Milner on page 72 of [Mil4]; to make the restriction precise, defining b to be unguarded in B if it occurs without an enclosing guard, a (set of mutually) recursive behaviour definition(s) is not allowed if there exists an infinite sequence $b_{i(1)}, b_{i(2)}, \ldots$ such that, for each j, $b_{i(j+1)}$ is unguarded in $b_{i(j)}$. An obvious additional requirement for correctness of sorts is that when b is defined by the clause $b \Leftarrow B$ then $L(B) \subseteq L(b)$. Behaviour identifiers which are defined satisfying these restrictions are said to be guardedly well-defined.

Before proceeding to the derivation rules which give the operational semantics for behaviour expressions it is necessary to introduce two more items of notation: B{E/x} is used to denote the behaviour B with the expression E substituted for all free occurrences of x (that is, occurrences which do not lie within the scope of any guard of the form x) and FV(B) represents the set of all variables which occur free in B.

DERIVATION RULES FOR ATOMIC ACTIONS

The operational semantics of behaviours in CCS is defined using derivation rules. A binary relation $\xrightarrow{\mu}$ is defined over programs (behaviour expressions with no free variables) for μ of the form λv (where v is a value appropriate to the label λ) or τ. Informally $B \xrightarrow{\mu} B'$ means that B can undergo a μ-experiment (or action) and subsequently behaves as B'. The relations are defined by structural induction on programs from the rules which follow. (The word "action" is used to refer to a single action as defined by these rules; a "derivation" may be a sequence of such actions.)

(i) NIL has no actions.

(ii) B_1+B_2 can behave either as B_1 or B_2:

$$\frac{B_1 \xrightarrow{\mu} B_1'}{B_1 + B_2 \xrightarrow{\mu} B_1'} \qquad \frac{B_2 \xrightarrow{\mu} B_2'}{B_1 + B_2 \xrightarrow{\mu} B_2'}$$

(The first expression means that from B_1 B_1' it is possible to infer that B_1+B_2 B_1'.)

(iii) $\mu.B$ can undergo an action appropriate to the type of guard μ and subsequently behave as B (with the relevant value substituted for free occurrences of x in the case where the guard is of the form αx):

$\alpha x.B \xrightarrow{\alpha v} B\{v/x\}$
$\bar{\alpha}E.B \xrightarrow{\bar{\alpha}v} B$ where the expression E evaluates to v
$\tau.B \xrightarrow{\tau} B$

(iv) $B_1|B_2$ can undergo any atomic action of either B_1 or B_2:

$$\frac{B_1 \xrightarrow{\mu} B_1'}{B_1|B_2 \xrightarrow{\mu} B_1'|B_2} \qquad \frac{B_2 \xrightarrow{\mu} B_2'}{B_1|B_2 \xrightarrow{\mu} B_1|B_2'}$$

Alternatively B_1 and B_2 may interact:

$$\frac{B_1 \xrightarrow{\lambda v} B_1',\ B_2 \xrightarrow{\bar{\lambda}v} B_2'}{B_1|B_2 \xrightarrow{\tau} B_1'|B_2'}$$

(This interaction cannot be observed by an agent external to $B_1|B_2$.)

(v) $B\backslash\alpha$ can undergo any action of B except α- or $\bar{\alpha}$-actions:

$$\frac{B \xrightarrow{\mu} B'}{B\backslash\alpha \xrightarrow{\mu} B'\backslash\alpha} \text{ if } name(\mu) \neq \alpha$$

(vi) B[S] can undergo any action of B with appropriate

relabelling:

$$\frac{B \xrightarrow{\nu} B'}{B[S] \xrightarrow{S\nu} B'[S]}$$

(vii) **if** E **then** B_1 **else** B_2 has the atomic actions of B_1 or B_2 according to the value of the boolean expression E:

$$\frac{B_1 \xrightarrow{\nu} B_1'}{\textbf{if } E \textbf{ then } B_1 \textbf{ else } B_2 \xrightarrow{\nu} B_1'} \text{ if E evaluates to true}$$

$$\frac{B_2 \xrightarrow{\nu} B_2'}{\textbf{if } E \textbf{ then } B_1 \textbf{ else } B_2 \xrightarrow{\nu} B_2'} \text{ if E evaluates to false}$$

(viii) If the behaviour identifier b has a defining clause $b(x_1,\ldots,x_{n(b)}) \Leftarrow B$ where $FV(B) \subseteq \{x_1,\ldots,x_{n(b)}\}$ then:

$$\frac{B\{v_1/x_1,\ldots,v_{n(b)}/x_{n(b)}\} \xrightarrow{\nu} B'}{b(E_1,\ldots,E_{n(b)}) \xrightarrow{\nu} B'}$$ where each E_i evaluates to v_i

Thus $b(E_1,\ldots,E_{n(b)})$ behaves as B with the appropriate values substituted for free occurences of the variables x_i.

From these rules it is possible to prove that if a behaviour B can undergo a $\lambda\nu$-action resulting in B' then λ is in the sort of B and furthermore the behaviour B' has the sort of B; this is stated in the following proposition (in which, as usual, B:L is used to mean that B has sort L). It should be noted that proofs are omitted throughout this chapter; each result stated will be followed by a reference to the appropriate page of [Mil4] where the proof, or an indication of the technique used, can be found.

Proposition 2.1

If a behaviour B has sort L and a derivation $B \xrightarrow{\lambda v} B'$ then $\lambda \in L$ and B':L; also, if $B \xrightarrow{\tau} B'$ then B':L. (p73)

This result is important in many proofs about the calculus; it is generally assumed implicitly rather than being referred to explicitly.

Since the derivation rules do not give meanings to behaviour expressions with free variables, it is not possible to define directly equivalences between such expressions; hence definitions of equivalences will be made initially with respect to programs, that is behaviour expressions without free variables, and subsequently extended to general behaviour expressions.

STRONG CONGRUENCE

Before defining observational equivalence on behaviours it is useful to consider first a stronger equivalence which is based directly on the derivation rules given in the preceding section. This equivalence is found to be a congruence and is hence known as strong congruence. As discussed earlier the derivation rules do not give a direct meaning to behaviour expressions containing free variables, hence the equivalence is defined initially with respect to programs. For two programs B and C to be equivalent, the necessary property is that if $B \xrightarrow{\mu} B'$ then C can undergo an action $C \xrightarrow{\mu} C'$ with B' and C' equivalent (and likewise with the roles of B and C interchanged). This cannot be used as a definition since it

"calls itself" recursively; hence it is necessary to define the equivalence, which is denoted by $\sim$, as the limit of a decreasing sequence of relations $\sim_k$. Hence:

Definition

$B \sim_0 C$ is true for all programs B and C;

$B \sim_{k+1} C$ holds for $k \geq 0$ if and only if for all $\mu \in \Lambda \times V \cup \{\tau\}$

(i) if $B \xrightarrow{\mu} B'$ then for some C', $C \xrightarrow{\mu} C'$ and $B' \sim_k C'$; and

(ii) if $C \xrightarrow{\mu} C'$ then for some B', $B \xrightarrow{\mu} B'$ and $B' \sim_k C'$.

$B \sim C$ if and only if $B \sim_k C$ for all $k \geq 0$.

It is easy to see that each $\sim_k$ is an equivalence relation and that $B \sim_{k+1} C$ implies $B \sim_k C$; it follows that $\sim$ is an equivalence relation. Two important properties follow:

Theorem 2.2

$\sim$ is a congruence relation. (p78)

Theorem 2.3

$B \sim C$ if and only if

(i) if $B \xrightarrow{\mu} B'$ then for some C', $C \xrightarrow{\mu} C'$ and $B' \sim C'$; and

(ii) if $C \xrightarrow{\mu} C'$ then for some B', $B \xrightarrow{\mu} B'$ and $B' \sim C'$. (p81)

The above theorem holds for recursively-defined behaviour identifiers only if they are guardedly well-defined, and provides some justification for the restriction introduced earlier that all such identifiers must satisfy this constraint.

Having defined strong congruence of programs and introduced some of its properties it is appropriate to list a series of equational laws of this equivalence.

Theorem 2.4

The following strong congruences of programs hold: (p75)

(1) $B_1 + B_2 \sim B_2 + B_1$

(2) $B_1 + (B_2 + B_3) \sim (B_1 + B_2) + B_3$

(3) $B + NIL \sim B$

(4) $B + B \sim B$

(5) $\alpha x.B \sim \alpha y.B\{y/x\}$ if y does not occur free in B

(6) $NIL\backslash\alpha \sim NIL$

(7) $(B_1 + B_2)\backslash\alpha \sim B_1\backslash\alpha + B_2\backslash\alpha$

(8) $(\mu.B)\backslash\alpha \sim \begin{cases} NIL \text{ if } \alpha = name(\mu) \\ \mu.B\backslash\alpha \text{ otherwise} \end{cases}$

(9) $NIL[S] \sim NIL$

(10) $(B_1 + B_2)[S] \sim B_1[S] + B_2[S]$

(11) $(\mu.B)[S] \sim S\mu.B[S]$

(12) if $b(x_1,\ldots,x_n) \Leftarrow B$ then
$b(v_1,\ldots,v_n) \sim B\{v_1/x_1,\ldots,v_n/x_n\}$

(13) if true then B_1 else $B_2 \sim B_1$

(14) if false then B_1 else $B_2 \sim B_2$

Milner in [Mil4] introduces the above as "direct equivalences" by defining another equivalence which is not needed here; as this equivalence is stronger than strong congruence the theorem in the above form follows immediately from Milner's result.

As the + operation is now known to be associative and commutative with respect to ~ it is possible to use the familiar notation of Σ to represent finite ambiguity. A sum $\hat{\Sigma}_{i\in I}\{B_i\}$ is said to be a sum of guards if each B_i is of the form $\mu.B_i'$, and each B_i can be referred to as a summand. Then:

Theorem 2.5

If two programs B and C are sums of guards then

$$
\begin{aligned}
B|C \sim\ & \Sigma\{\mu.(B'|C) : \mu.B' \text{ a summand of } B\} \\
& + \Sigma\{\mu.(B|C') : \mu.C' \text{ a summand of } C\} \\
& + \Sigma\{\tau.(B'\{v/x\}|C') : \alpha x.B' \text{ a summand of } B \text{ and } \bar{\alpha}v.C' \text{ a summand of } C\} \\
& + \Sigma\{\tau.(B'|C'\{v/x\}) : \bar{\alpha}v.B' \text{ a summand of } B \text{ and } \alpha x.C' \text{ a summand of } C\}. \text{ (p75)}
\end{aligned}
$$

This result was also stated by Milner with respect to direct equivalence. The following, are however, not direct equivalences.

Theorem 2.6

The following strong congruences hold for programs: (p79)

(1) $B_1|B_2 \sim B_2|B_1$

(2) $B_1|(B_2|B_3) \sim (B_1|B_2)|B_3$

(3) $B|NIL \sim B$

(4) $B\backslash\alpha \sim B$ $\quad(\alpha, \bar{\alpha} \notin L(B))$

(5) $B\backslash\alpha\backslash\beta \sim B\backslash\beta\backslash\alpha$

(6) $(B_1|B_2)\backslash\alpha \sim B_1\backslash\alpha|B_2\backslash\alpha$ $\quad(\alpha, \bar{\alpha} \notin L(B_1) \cap \overline{L(B_2)})$

(7) $B[I] \sim B$ (where I is the identity relabelling)

(8) $B[S] \sim B[S']$ if $S\lambda = S'\lambda$ for all $\lambda \in L(B)$

(9) $B[S][S'] \sim B[S' \circ S]$ (where $\circ$ denotes composition)

(10) $B[S]\backslash\beta \sim B\backslash\alpha[S]$ (where $\beta = name(S\alpha)$)

(11) $(B_1 | B_2)[S] \sim B_1[S] | B_2[S]$.

In view of (5) above there is no ambiguity in using $B\backslash A$, where A is the set $\{\alpha_1,\ldots,\alpha_n\}$, to represent $B\backslash\alpha_1\ldots\backslash\alpha_n$.

All the definitions and properties introduced so far for strong congruence have been with respect to programs rather than arbitrary behaviour expressions. The definition is extended in the obvious manner: two behaviours B_1 and B_2 are strongly congruent if and only if they are congruent for any substitution of values to their free variables. The following theorem extends the results above in view of this definition:

Theorem 2.7

Strong congruence is a congruence over behaviour expressions. The results of theorems 2.4 through 2.6 hold for arbitrary expressions subject to the following modifications:

(i) In (12) of theorem 2.4 v should be replaced by E (an arbitrary expression.

(ii) In theorem 2.5 v should be replaced by E; additionally it is necessary to impose the condition that in the first sum of the right-hand-side no free variable of C is bound by μ and likewise in the second sum no free variable of B is thus bound. (p82)

From the four preceding results can be deduced a property known as the expansion theorem, which is widely used in proofs of equivalence of behaviours.

Theorem 2.8 (Expansion Theorem)

If $B \equiv (B_1|\ldots|B_n)\backslash A$, where each B_i is a sum of guards, then

$$B \sim \{\mu.((B_1|\ldots|B_i'|\ldots|B_m)\backslash A) : \mu.B_i' \text{ a summand of } B_i, \text{name}(\mu) \notin A\}$$
$$+ \{\tau.((B_1|\ldots|B_i'\{E/x\}|\ldots|B_j'|\ldots|B_m)\backslash A) : \alpha x.B_i' \text{ a summand of } B_i \text{ and } \bar{\alpha}E.C' \text{ a summand of } B_j \text{ for } i \neq j\}$$

provided that in the first term no free variable in any B_k for $k \neq i$ is bound by μ. (p82)

All the major properties of strong congruence have been listed; it remains to define observation equivalence, and after stating that this is weaker than strong congruence it will follow that the results of this section are also properties of observation equivalence.

OBSERVATION EQUIVALENCE AND CONGRUENCE

Observation equivalence is defined in a similar manner to strong congruence; however it allows unobservable τ-actions to be absorbed into actions and also takes into account multiple-length derivations as well as atomic actions. Before defining the equivalence it is necessary to introduce notation to represent such derivations. Firstly a sequence of actions can be represented in a single derivation by $B \xrightarrow{\mu_1 \cdots \mu_n} B'$; a derivation of this form means that there is some sequence of behaviours B_i such that B_0 is B, B_n is B' and for $i \in \{1,\ldots,n\}$ $B_{i-1} \xrightarrow{\mu_i} B_i$. It is also possible to write $B \xrightarrow{\varepsilon} B'$ (where ε is the empty string) which means that B is B' (as no action is needed to obtain one from the other). To allow the absorption of τ-experiments $B \xrightarrow{\tau^n} B'$ (for $n \geq 0$) may be abbreviated to $B \overset{\varepsilon}{\Rightarrow} B'$, and $B \xrightarrow{\tau^m \mu \tau^n} B'$ (for $m,n \geq 0$) to $B \overset{\mu}{\Rightarrow} B'$. Finally, for strings $s \in (A \times V)^*$, $B \overset{s}{\Rightarrow} B'$ is

defined in the obvious way so that if $s = \mu_1 \cdots \mu_n$ then $B \overset{\mu_1}{\Longrightarrow} B_1 \cdots B_{n-1} \overset{\mu_n}{\Longrightarrow} B'$. (As a consequence of these definitions it should be noted that $B \overset{\tau}{\Rightarrow} B'$ means $B \xrightarrow{\tau^n} B'$ for $n \geq 1$ whereas $B \overset{\varepsilon}{\Rightarrow} B'$ means $B \xrightarrow{\tau^n} B'$ for $n \geq 0$.)

Having introduced this notation observation equivalence is defined analogously to strong congruence using $\overset{s}{\Rightarrow}$ instead of $\xrightarrow{\mu}$.

Definition

$B \approx_0 C$ is true for all programs B and C;

$B \approx_{k+1} C$ holds for $k \geq 0$ if and only if for all $s \in (\Lambda \times V)^*$

(i) if $B \overset{s}{\Rightarrow} B'$ then for some C', $C \overset{s}{\Rightarrow} C'$ and $B' \approx_k C'$; and

(ii) if $C \overset{s}{\Rightarrow} C'$ then for some B', $B \overset{s}{\Rightarrow} B'$ and $B' \approx_k C'$.

$B \approx C$ if and only if $B \approx_k C$ for all $k \geq 0$.

It should be remembered that this definition applies only to programs, without free variables; the extension to arbitrary behaviour expressions is made in the same way as for strong congruence.

An immediate property of observation equivalence is the following, which does not hold for strong congruence:

Proposition 2.9

$B \approx \tau.B$ for all behaviours B. (pp100,102)

In order that the strong congruences already stated can be used as observation equivalences the following result is necessary.

Theorem 2.10

For all behaviours B and C, $B \sim C$ implies $B \approx C$. (p101)

Observation equivalence is not a congruence: for example $NIL \approx \tau.NIL$ but it is not true that $\alpha.NIL + NIL \approx \alpha.NIL + \tau.NIL$. However it is a congruence for all behaviour operations except +:

Theorem 2.11

For any behaviour expressions B and C, $B \approx C$ implies:

(i) $\alpha v.B \approx \alpha v.C$
(ii) $\tau.B \approx \tau.C$
(iii) $B|D \approx C|D$
(iv) $B\backslash\alpha \approx C\backslash\alpha$
(v) $B[S] \approx C[S]$

Also, if $B\{v/x\} \approx C\{v/x\}$ for all v then $\tilde{\alpha}x.B \approx \tilde{\alpha}x.C$. (p101)

Since observation equivalence is not a congruence, and it is derirable at times to be able to work with a congruence, it is useful to define observation congruence as follows.

Definition

Two behaviour expressions B and C are said to be observationally congruent (written $B \approx^{c} C$) if and only if for every context $F[\ ]$, $F[B] \approx F[C]$.

A context is understood to be a behaviour expression with a "hole" and may be regarded as a functional. This definition has

the following important properties:

Theorem 2.12

$\approx^C$ is a congruence, and is the weakest congruence stronger than $\approx$. (p103)

Theorem 2.13

$B \approx^C C$ if and only if, for all D, $B + D \approx C + D$. (p104)

A behaviour B is said to be stable if it can undergo no derivations of the form $B \xrightarrow{\tau} B'$. A related concept is that of being rigid; a behaviour is rigid if all of its derivatives (including itself, which may now be regarded as an ε-derivative) are stable.

Proposition 2.14

If B and C are stable and $B \approx C$ then $B \approx^C C$. (p105)

An immediate corollary of this result is that if $B \approx C$ then, for any guard μ not equal to τ, $\mu.B \approx^C \mu.C$. In fact this result also holds for $\mu = \tau$:

Proposition 2.15

For any guard μ, $B \approx C$ implies $\mu.B \approx^C \mu.C$. (p105)

All of the laws of strong congruence hold for observation congruence since the latter is weaker than the former; as observation congruence allows unobservable actions to be absorbed there are some additional laws which can be introduced involving expressions containing the τ guard:

Theorem 2.16

The following observation congruences of behaviour expressions hold: (pp106-107)

(1) $\mu.\tau.B \approx^C \mu.B$

(2) $B + \tau.B \approx^C \tau.B$

(3) $\mu.(B + \tau.C) + \mu.C \approx^C \mu.(B + \tau.C)$

(4) $B + \tau.(B + C) \approx^C \tau.(B + C)$

The last law above is actually a corollary of the second (together with the laws of + introduced earlier) but is included in the theorem as it is often more useful in practise. It is known as the absorption rule.

The notation = is frequently used to represent observation congruence; hence to avoid any ambiguity in subsequent chapters $\equiv$ is employed to indicate identical structure of agents.

The properties of $\approx$ and $\approx^C$ introduced, together with the "laws" introduced earlier for strong congruence, which hold for $\approx$ and $\approx^C$, provide a useful framework for proving the equivalence of behaviours in CCS. The remainder of this thesis aims to introduce "larger-scale" properties which can help to eliminate much of the paperwork involved in the repeated application of these results.

PURE CCS

The restricted subcalculus of CCS involving only pure synchronisation and no value-passing or parameterised behaviour expressions is referred to as pure CCS. In this subcalculus all guards are of the form α, $\bar{\alpha}$ or τ. Since variables do not occur the only conditional expressions that can be constructed take the form

if true then B_1 else B_2 or

if false then B_1 else B_2

and as these can be replaced by B_1 and B_2 respectively without any change of meaning it is reasonable to omit conditional expressions from the syntax of the pure calculus.

3 STRONG CONFLUENCE AND DETERMINACY

INTRODUCTION

The concepts of confluence and determinacy were introduced in [Mil4] in order to encapsulate the property of determinism (in some intuitive sense) from the observer's viewpoint, which is displayed by many programs written in CCS. Milner demonstrated that strongly confluent behaviours satisfy a property enabling τ-actions to be ignored in proofs of equivalence, considerably shortening such proofs in many cases. The concept of confleunce is not unique to CCS; similar properties have been studied by Huet [Hue] for reduction relations and occur in a more familiar role in the λ-calculus as the Church-Rosser theorem [HLS].

From the definition given by Milner, a strongly confluent behaviour, A, in CCS has to satisfy two conditions: firstly, if $A \xrightarrow{\mu} B$ and $A \xrightarrow{\nu} C$ then either $\mu = \nu$ and $B \sim C$ or $B \xrightarrow{\nu} D$ and $C \xrightarrow{\mu} E$ for behaviours D and E such that $D \sim E$, and, secondly, any derivatives of A must also be strongly confluent. Comparing this with Huet's study it is seen immediately that the presence of labels accompanying the derivations complicates the definition; however the underlying concept is the same. Huet's definition of confluence is made with respect to a system of terms with a single reduction relation, whereas in CCS confluence is defined on single terms (behaviours) in the presence of a set of relations. This difference is not as significant as it may appear, since the effect of the second condition referred to in the informal definition above is that a subcalculus of CCS is being studied rather than a single behaviour; the statement "A is confluent" actually means that the system comprising the behaviour A and all its derivatives is confluent. Analogies can be drawn between applications of the Strong Confluence Theorem of CCS, which states that if a behaviour A is confluent and $A \xrightarrow{\tau} B$ then $A \sim B$, to proofs of observational equivalence and Huet's applications of confluent

reductions to term-rewriting systems.

The Church-Rosser property of the λ-calculus differs from Huet's confluence since it involves more than one reduction relation; the principal difference between this form of confluence of the λ-calculus and strong confluence in CCS is that the former does not imply the "single-step" nature of the definition of the latter.

Another property bearing some relation to confluence is freedom from conflict in net theory [GLT]. To see this similarity suppose an agent A is confluent and has derivations $A \xrightarrow{\mu} B$ and $A \xrightarrow{\nu} C$. Then the performance of a μ-experiment does not interfere in any way with the capability to perform a ν-experiment, so these two experiments exhibit some form of causal independence, and can be compared with transitions in net theory which are conflict-free.

Milner's study of confluence and determinacy in [Mil4] was restricted to the limited case of "pure" CCS, involving only synchronisation and not allowing variables and the passing of parameters. Here the definitions will be extended to the full CCS incorporating value-communication, and their properties will be investigated providing results leading to a Strong Confluence Theorem analogous to that of Milner. In addition it will be shown that certain operations preserve strong confluence and determinacy and these will be used to define a subcalculus in which all behaviours are confluent, enabling the Strong Confluence Theorem to be used freely in proofs. Examples of its application will be studied to demonstrate how it can reduce the length of proofs of observational equivalence.

Notation

Throughout the definitions and proofs in this section A,B,C... will be used to represent programs, but U,V,W... will signify any

behaviour expression, possibly containing free variables.

λ will, as in the previous chapter, represent a label in Λ $(=\Delta\cup\bar{\Delta})$, whereas $\alpha,\beta,\gamma,\ldots$ will refer to positive labels in Δ.

μ and ν will range over the set $(\Lambda\times V)\cup\{\tau\}$ (where V is the set of values that can be passed).

σ will represent a string of elements of $(\Lambda\times V)\cup\{\tau\}$ while s will represent a string of elements of $\Lambda\times V$.

STRONG CONFLUENCE

As stated in the introduction, the definition of strong confluence given by Milner for pure CCS implies essentially that a behaviour A is strongly confluent if it satisfies two conditions: firstly, that if $A\xrightarrow{\mu}B$ and $A\xrightarrow{\nu}C$ then either $\mu=\nu$ and $B\sim C$, or $B\xrightarrow{\nu}D$ and $C\xrightarrow{\mu}E$ for some D and E such that $D\sim E$, and, secondly, that any derivative of A is also strongly confluent. This is not a definition as such as it makes use recursively of the property it attempts to define; hence it is necessary to use a sequence of definitions of k-confluence and take the intersection over $k\geq 0$.

Milner's definition is not quite appropriate for the full CCS with value-communication; consider a few simple examples. Firstly, let A be the simple adder defined by:

$$A \Leftarrow \alpha x.\beta y.\bar{\gamma}(x+y).NIL$$

(and assume that the set of values that x can take is a finite subset of the integers). Then

$$A\xrightarrow{\alpha 7}\beta y.\bar{\gamma}(7+y).NIL$$

$$\text{and } A\xrightarrow{\alpha 5}\beta y.\bar{\gamma}(5+y).NIL.$$

Under the original definition this is clearly not confluent as the two right-hand sides of these derivations cannot undergo $\alpha 5$- and $\alpha 7$-experiments. However we should want a behaviour such as A to be confluent if our definition is to encompass any non-trivial cases of value-communication. Hence we need to include an extra clause to allow for the case of $A\xrightarrow{\alpha m}B$ and $A\xrightarrow{\alpha n}C$ where B and C are equivalent except by being parameterised by m and n

respectively. However this does not apply in the case of $\bar{\alpha}m$ and $\bar{\alpha}n$ derivations; for example, let A' be defined by:

$$A' \Leftarrow \bar{\alpha}5.NIL + \bar{\alpha}7.NIL$$

We would not expect A' to be confluent as it can offer two totally different results to its environment. Bearing these examples in mind and using a sequence of properties of k-confluence for $k \geq 0$ to avoid problems of recursion the following definitions are made.

Definitions

A program A is always strongly 0-confluent (abbreviated to SC_0).

A is strongly (k+1)-confluent (SC_{k+1}) for $k \geq 0$ if and only if:

(i) Whenever $A \xrightarrow{\mu} B$ and $A \xrightarrow{\nu} C$ then

either $B \xrightarrow{\nu} D$ and $C \xrightarrow{\mu} E$ for some D and E such that $D \sim E$,

or $\mu = \nu$ and $B \sim C$,

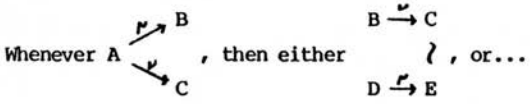
or $\mu = \alpha m$ and $\nu = \bar{\alpha}n$ for some positive label α and values m and n, with $B \sim U\{m/x\}$ and $C \sim U\{n/x\}$ for some behaviour U.

(ii) $A \xrightarrow{\mu} B$ implies that B is strongly k-confluent.

A is strongly confluent (abbreviated to SC) if and only if A is strongly k-confluent for all $k \geq 0$.

There follows an investigation of certain properties of these definitions culminating in the Strong Confluence Theorem, a result which states that unobservable actions cannot affect the capabilities of a confluent program (that is, all τ-derivatives are equivalent to the program itself). The fact that we are able to prove this theorem helps to justify the form of the above definitions. Applying this result often enables unobservable actions to be ignored in proofs of equivalence. It is convenient

to use diagrams to illustrate the derivations involved in the proofs in this section. For example, part (i) of the definition of k-confluence above can be expressed as:



Diagrammatic representations such as this will prove useful as an aid to understanding and also make the proofs more readable.

In the proofs that follow strong confluence will often be referred to simply as "confluence"; since no other form of confluence has yet been defined there should be no confusion. The abbreviated forms SC and SC_k will also be used frequently when convenient.

Proposition 3.1

A program A is strongly confluent if and only if condition (i) of the above definition holds and whenever $A \xrightarrow{r} B$ then B is also strongly confluent.

Proof

This result follows immediately from the definition by a simple induction.

This proposition provides a recursive treatment of strong confluence, which will often be useful in the following proofs. It could not have been used as a definition of confluence because of its recursive nature.

Strong confluence is not preserved by $\simeq$; an example analogous to that given in [Mil4] will demonstrate this:

$$\alpha x.\beta y.NIL + \beta y.\alpha x.NIL \simeq \alpha x.\beta y.NIL + \beta y.\tau.\alpha x.NIL$$

Here the first behaviour is confluent, but the second is not, due to the τ-action. However the next result shows that confluence is preserved by strong congruence, $\sim$.

Proposition 3.2

If a program A is strongly confluent and $A \sim A'$ then A' is also strongly confluent.

Proof

We prove that the result holds for strong k-confluence for all $k \geq 0$ by induction on k. The result is trivially true for k=0; for the inductive step we assume that the result holds for k-confluence, and prove that it holds for (k+1)-confluence. So suppose A is SC_{k+1} and $A \sim A'$. To prove that A' is SC_{k+1} it has to be shown that the two clauses in the definition are satisfied. The second clause follows trivially from the inductive hypothesis and theorem 1.3. It remains to prove that A' satisfies condition (i) of the definition. So suppose that

$$A' \xrightarrow{\mu} B' \quad , \quad A' \xrightarrow{\nu} C'$$

. Then

$$A \xrightarrow{\mu} B \sim B' \quad , \quad A \xrightarrow{\nu} C \sim C'$$

, for some B and C.

A is SC_{k+1} so from the definition there are three cases to consider:

(i) $B \xrightarrow{\nu} D$ $\quad B' \xrightarrow{\nu} D' \sim D$ $\quad B' \xrightarrow{\nu} D'$

$\quad\quad \wr$, hence $\quad\quad \wr$, so $\quad\quad \wr$ as required.

$C \xrightarrow{\nu} E$ $\quad C' \xrightarrow{\nu} E' \sim E$ $\quad C' \xrightarrow{\nu} E'$

(ii) $\nu=\mu$ and $B \sim C$. Then $B' \sim B \sim C \sim C'$.

(iii) $\mu=\alpha m$, $\nu=\alpha n$, $B \sim U\{m/x\}$ and $C \sim U\{n/x\}$. Then $B' \sim U\{m/x\}$ and $C' \sim U\{n/x\}$.

In all three cases A' satisfies the first clause of the definition of (k+1)-confluence, hence A' has been shown to be SC_{k+1}, completing the inductive step of the proof.

The two results 3.1 and 3.2 proved so far are directly analogous to 10.1 and 10.2 of [Mil4]. The following lemma is weaker than Milner's 10.3, but it will satisfy the requirements of the proof of the Strong Confluence Theorem. It is possible to prove a result along the lines of 10.3, but the statement of such a lemma would be rather lengthy and the proof, although routine, somewhat untidy. (In fact a lemma similar to the one below would have been adequate for Milner's requirements in the pure CCS, but there the stronger result required little extra effort.)

Lemma 3.3

If a program A is strongly confluent and $A \xrightarrow{\tau} B$, $A \xrightarrow{\mu_1 \cdots \mu_n} C$ then either

$B \xrightarrow{\mu_1 \cdots \mu_n} D$ $\quad\quad$ $B \xrightarrow{\mu_1 \cdots \mu_{i-1} \mu_{i+1} \cdots \mu_n} D$

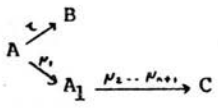
$\quad\quad \wr$ or $\mu_i = \tau$ for some i and $\quad\quad \wr$.

$C \xrightarrow{\tau} E$ $\quad\quad$ C

Proof

By induction on the length n of the derivation $A \rightarrow C$.

For n=1 the result is immediate from proposition 3.1.
For the inductive step we assume the result holds for derivations of length at most n and prove it for length n+1.
We have:

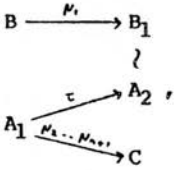


(splitting the derivation $A \to C$ into two parts).

Now by proposition 3.1 either:

(i) $\mu_1 = \tau$ and $A_1 \sim B$, in which case $B \xrightarrow{\mu_2 \cdots \mu_{n+1}} D \sim C$ and the result is proved; or

(ii)



in which case A_1 is confluent by proposition 3.1, allowing the application of the inductive hypothesis to the lower half of the diagram. Hence

either $\begin{matrix} A_2 \xrightarrow{\mu_2 \cdots \mu_{n+1}} D_1 \\ C \xrightarrow{\tau} E \end{matrix}$ when as $A_2 \sim B_1$ we have

$B_1 \xrightarrow{\mu_2 \cdots \mu_{n+1}} D \sim E$, so $\begin{matrix} B \xrightarrow{\mu_1 \cdots \mu_{n+1}} D \\ C \xrightarrow{\tau} E \end{matrix}$, as required;

or $\mu_i = \tau$ for some i and $A_2 \xrightarrow{\mu_2 \cdots \mu_{i-1} \mu_{i+1} \cdots \mu_{n+1}} D_1 \sim C$,
when as $A_2 \sim B_1$ we have $B_1 \xrightarrow{\mu_2 \cdots \mu_{i-1} \mu_{i+1} \cdots \mu_{n+1}} D \sim C$,

$$B \xrightarrow{\mu_1 \cdots \mu_{i-1} \mu_{i+1} \cdots \mu_{n+1}} D$$

hence $\quad C \quad$, as required.

This shows that the result holds for derivations of length n+1, completing the inductive step of the proof.

Having completed the preliminary results, we are now ready to state and prove the Strong Confluence Theorem, which states that any τ-derivative of a confluent program is observationally equivalent to the program itself.

Theorem 3.4 (Strong Confluence Theorem)

If a program A is strongly confluent and $A \xrightarrow{\tau} B$ then $A \approx B$.

Proof

It is necessary to show that $A \approx_k B$ for all $k \geq 0$. This will be done by induction on k. The result holds trivially for k=0. For the inductive step we assume that the result holds for k (for all confluent A) and prove that it holds for k+1:

(i) Suppose $B \stackrel{s}{\Rightarrow} B'$. Then $A \xrightarrow{\tau} B$, so $A \stackrel{s}{\Rightarrow} B'$ (and of course $B' \approx_k B'$)

(ii) Conversely suppose $A \stackrel{s}{\Rightarrow} A'$. Then $A \xrightarrow{\sigma} A'$ for some σ such that s is the string σ with τ's removed. As $A \xrightarrow{\tau} B$ also, lemma 3.3 can be applied giving

$$\begin{array}{lll} & B & \xrightarrow{\sigma} D \\ \text{either} & & \downarrow \\ & A' & \xrightarrow{\tau} E \end{array}$$

in which case $B \stackrel{s}{\Rightarrow} D$. By the inductive hypothesis we have $A' \approx_k E$, and since $D \sim E$ this gives $A' \approx_k D$;

$$B \xrightarrow{\sigma_1,\sigma_2} D$$

or $\sigma=\sigma_1\tau\sigma_2$ and $\quad\wr$,

$$A'$$

when $B \stackrel{s}{\Rightarrow} D$ and $A' \approx_k D$.

In both cases we have found D such that $B \stackrel{s}{\Rightarrow} D$ and $A' \approx_k D$, completing the proof that $A \approx_{k+1} D$

Examples of the applications of this result to proofs of equivalence will be given later, but first we will introduce a related concept, determinacy, and demonstrate that certain operations in CCS preserve confluence and determinacy, enabling the construction of a subcalculus in which all behaviours are confluent.

STRONG DETERMINACY

As in the case of strong confluence the definition of strong determinacy for the full CCS is based on that given by Milner in [Mil14]. Once again it is found that certain modifications need to be made to accommodate the possibilities of a determinate behaviour being able to accept different values as "input" via a single label, while ensuring that the "output" from a determinate program is intuitively "deterministic"; for example we should expect $\alpha x.\bar{\beta}x.NIL$ to be determinate, but not $\alpha x.(\bar{\beta}3.NIL+\bar{\beta}4.NIL)$.

Milner's definition of strong determinacy for the pure calculus made use of concepts of k-determinacy for $k \geq 0$ in a similar way to the confluence definition. Furthermore, λ-determinacy was defined for each label λ. We are able to take advantage of this by treating the definition of α-determinacy differently from that of $\bar{\alpha}$-determinacy, hence being able to deal with the distinct requirements of the offering and accepting of values separately.

The non-recursive part of Milner's definition of a λ-determinate behaviour A required that whenever $A \xrightarrow{\lambda} B$ and $A \xrightarrow{\lambda} C$ then $B \sim C$. Here we also have to consider the values being passed along with the labels. One would expect a program such as $\alpha x.\bar{\beta} x.NIL$ to be determinate so in defining α-determinacy we will need to allow $A \xrightarrow{\alpha m} B$ and $A \xrightarrow{\alpha n} C$ where $m \neq n$. This presents no problems; the condition we impose is the same as in the definition of strong confluence, that is that $B \sim U\{m/x\}$ and $C \sim U\{n/x\}$ for some behaviour U. In the case of the negative guard it would clearly be undesirable to admit a program such as $\bar{\alpha} 3.NIL + \bar{\alpha} 4.NIL$ as α-determinate; hence when $A \xrightarrow{\bar{\alpha} m} B$ and $A \xrightarrow{\bar{\alpha} n} C$ it will be necessary to insist that $m=n$ and $B \sim C$.

Definitions

For any $\lambda \in \Lambda$, any program A is strongly λ-0-determinate (abbreviated to λ-SD_0).

A is α-SD_{k+1} (for $k > 0$) if and only if:
(i) Whenever $A \xrightarrow{\alpha m} B$ and $A \xrightarrow{\alpha n} C$ then $B \sim U\{m/x\}$ and $C \sim U\{n/x\}$ for some behaviour U.
(ii) $A \xrightarrow{\mu} B$ implies that B is α-SD_k.

A is $\bar{\alpha}$-SD_{k+1} (for $k > 0$) if and only if:
(i) If $A \xrightarrow{\bar{\alpha} m} B$ and $A \xrightarrow{\bar{\alpha} n} C$ then $m=n$ and $B \sim C$.
(ii) $A \xrightarrow{\mu} B$ implies that B is $\bar{\alpha}$-SD_k.

A is strongly λ-determinate (λ-SD) if and only if A is λ-SD_k for all $k > 0$.

A is strongly determinate (SD) if and only if A is λ-SD for all

We can define strong k-determinacy in the obvious way by substituting SD_k for SD throughout the last clause of the above definitions. As with strong confluence we shall occasionally omit the word "strong" in proofs to aid readability.

Strong determinacy has some properties similar to strong confluence; propositions analogous to 3.1 and 3.2 follow; these results will be used in proving that the subcalculus we subsequently define is confluent and determinate.

Proposition 3.5

A program A is strongly determinate if and only if:

(i) whenever $A \xrightarrow{\alpha m} B$ and $A \xrightarrow{\alpha n} C$ then $B \sim U\{m/x\}$ and $C \sim U\{n/x\}$ for some U;

(ii) whenever $A \xrightarrow{\overline{\alpha} m} B$ and $A \xrightarrow{\overline{\alpha} n} C$ then $m=n$ and $B \sim C$; and

(iii) if $A \xrightarrow{\tau} B$ then B is SD.

Proof

By simple induction from the definition.

This result will prove useful as a recursive characterisation of strong determinacy, as proposition 3.1 did for strong confluence. It is easy to see how to formulate a similar result for λ-determinacy.

Observational equivalence does not preserve strong determinacy; consider the following example:

$$\alpha x.(\overline{\beta} x.NIL + \tau.NIL) + \alpha x.NIL \simeq \alpha x.(\overline{\beta} x.NIL + \tau.NIL)$$

This equivalence can be deduced from the absorption rule (from theorem 2.16 in the previous chapter). It is not difficult to

demonstrate that the right-hand-side is strongly determinate, but clearly the behaviour on the left is not α-determinate. However, as with confluence it can be shown that strong equivalence ($\sim$) does preserve SD.

Proposition 3.6

If $A \sim A'$ and A is strongly determinate then so is A'.

Proof

The same method is used as in the proof of proposition 3.2, and it is found that the work involved is slightly simpler.

It has been shown that strong equivalence preserves SC and SD. So far these concepts have been treated separately; however in the next section we shall be interested in whether a behaviour is both strongly confluent and determinate so it will be useful to introduce the abbrevaition SCD for this property.

We have now presented sufficient properties about strong confluence and determinacy to be enable their use in proofs of equivalence in CCS; the next step is to present a confluent and determinate subcalculus in which these results can be freely applied. Examples will be given to show how this can be used in proof techniques.

A CONFLUENT DETERMINATE SUBCALCULUS

In order to find a non-trivial confluent determinate subcalculus of CCS it is necessary to investigate what operations on behaviours preserve strong confluence and determinacy. We commence by showing using a simple example that + and | do not

preserve SCD. Let A = αx.NIL and B = αx.$\bar{\alpha}$x.NIL; these are clearly SCD but A+B is not (since it is not α-SD); furthermore A|B is not SCD (as it is not α-SD either, for example there are two different α0-derivations).

We now proceed to show that restriction, relabelling and guarding do preserve SCD, and along with the trivial result that NIL is SCD, this gives us a simple confluent determinate subcalculus (which is too trivial to be of much practical use as it admits no interleaving).

Proposition 3.7

If a program A is strongly confluent and determinate then so is A\α.

Proof

The first step is to prove that A\$\alpha$ is SC. This is done by using induction on k to show that A\$\alpha$ is SC_k for all $k \geq 0$. The same technique is then used to show that A\$\alpha$ is λ-SD for $\lambda \notin \{\alpha, \bar{\alpha}\}$. The result that A\$\alpha$ is α-SD and $\bar{\alpha}$-SD is trivial since the behaviour and its derivatives can never perform an α- or $\bar{\alpha}$-action.

Proposition 3.8

If a program A is strongly confluent and determinate then so is A[β/α]

Proof

Similar to 3.7.

Proposition 3.9

If a program A ia strongly confluent and determinate then so is $\nu.A$.

Proof

This result follows trivially from propositions 3.1 and 3.5. A stronger result will be proved in proposition 3.12.

The three propositions above tell us (not surprisingly) that the subcalculus built from inaction (NIL), guarding, relabelling and restriction is confluent and determinate. This result is of no use in proofs of equivalence since they are already trivial within this subcalculus. We need to find other operations that preserve confluence and determinacy to allow at least a restricted use of composition and summation in the derived calculus that will be produced.

Recalling the example given to show that | failed to preserve confluence and determinacy, it is not difficult to see that the reason for this failure was that there was an overlap between the sorts of the behaviours A and B. To eliminate this we can impose a restriction that these sorts must be disjoint. Now consider a similar example with $A = \bar{\alpha}5.NIL$ and $B = \alpha x.\bar{\beta}x.NIL$; here the sort of A is $\{\bar{\alpha}\}$ and the sort of B is $\{\alpha,\bar{\beta}\}$ so the sorts are disjoint. However $A|B \xrightarrow{\bar{\alpha}5} NIL|B$ and $A|B \xrightarrow{\tau} \beta 5.NIL$, and this fails to satisfy the conditions for confluence. The problem here is that the $\bar{\alpha}5$-action of A may be observed either externally or by B; to avoid this we limit the composition in our subcalculus to $(A|B)\backslash L$, where L is the intersection of the sort of A and the co-sort of B. This can be formally defined as follows.

Definition

Let U_1 have sort L_1 and U_2 have sort L_2, with $L_1 \cap L_2 = \emptyset$. Then $U_1 \| U_2$ is defined to be $(U_1 | U_2) \backslash L$ where $L = \text{names}(L_1 \cap \overline{L_2})$. This operation is known as restricted disjoint composition (abbreviated to rd-composition).

We now proceed to show that these restrictions are sufficient to ensure that this restricted operation preserves strong confluence and determinacy.

Proposition 3.10

If two programs A_1 and A_2 with disjoint sorts L_1 and L_2 respectively are strongly confluent and determinate then so is $A_1 \| A_2$.

Proof

(We shall freely make use of the knowledge that $\sim$ is a congruence in this proof.)
We prove by induction that the result holds for SCD_k for $k \geq 0$. For the inductive step assume that A_1 and A_2 are SCD_{k+1} and that the result holds for SCD_k. It is necessary to show that $A_1 \| A_2$ is SC_{k+1} and SD_{k+1}; we deal with the confluence first, commencing by proving that the first clause of the definition is satisfied. So suppose:

$A_1 \| A_2 \overset{r}{\rightarrow} B$, $A_1 \| A_2 \overset{s}{\rightarrow} C$. We consider the possible cases that can occur:

(i) B is of the form $B_1 \| A_2$ and C is $A_1 \| C_2$, with $A_1 \overset{r}{\rightarrow} B_1$ and $A_2 \overset{s}{\rightarrow} C_2$. Then

$$B_1 \| A_2 \xrightarrow{\nu} B_1 \| C_2 \text{ , as required.}$$
$$A_1 \| C_2 \xrightarrow{\mu}$$

(ii) B is $B_1 \| A_2$ and C is $C_1 \| A_2$. Then we have

$$A_1 \overset{\mu}{\rightarrow} B_1 \quad , \text{ and we use the confluence of } A_1 \text{ to give}$$
$$A_1 \overset{\nu}{\rightarrow} C_1$$

$$\underline{\text{either}} \quad \begin{array}{ccc} B_1 & \xrightarrow{\nu} & D_1 \\ & \wr & \\ C_1 & \xrightarrow{\mu} & E_1 \end{array} \text{ , when } \quad \begin{array}{ccc} B_1 \| A_2 & \xrightarrow{\nu} & D_1 \| A_2 \\ & \wr & \\ C_1 \| A_2 & \xrightarrow{\mu} & E_1 \| A_2 \end{array} ;$$

$\underline{\text{or}}$ $\mu = \nu$ and $B_1 \sim C_1$, in which case $B \sim C$;

$\underline{\text{or}}$ $\mu = \alpha m$, $\nu = \alpha n$, $B_1 \sim U\{m/x\}$ and $C_1 \sim U\{n/x\}$. Then (as A_2 has no free variables) we have $B_1 \| A_2 \sim (U \| A_2)\{m/x\}$ and $C_1 \| A_2 \sim (U \| A_2)\{n/x\}$.

In all three cases the confluence definition is satisfied.

(iii) B is $B_1 \| B_2$ with $\mu = \tau$ and C is $C_1 \| A_2$. Here we use the confluence of A_1. There are two possibilities according to how the derivation $A_1 \| A_2 \xrightarrow{\tau} B_1 \| B_2$ was caused,

$$\underline{\text{either}} \; A_1 \overset{\alpha m}{\rightarrow} B_1 \quad , \text{ with } A_2 \xrightarrow{\overline{\alpha} m} B_2 \text{ and } \alpha \in L_1 \cap \overline{L_2},$$
$$A_1 \overset{\nu}{\rightarrow} C_1$$

when we cannot have $\nu = \alpha n$ since α is not in the sort of $A_1 \| A_2$. Hence the confluence of A_1 gives

$$\begin{array}{ccc} B_1 & \xrightarrow{\nu} & D_1 \\ & \wr & \\ C_1 & \xrightarrow{\alpha m} & E_1 \end{array} \text{ , so } \quad \begin{array}{ccc} C_1 \| A_2 & \xrightarrow{\tau} & E_1 \| B_2 \\ & \wr & \\ B_1 \| B_2 & \xrightarrow{\nu} & D_1 \| B_2 \end{array} ;$$

$$\text{or}\quad A_1 \begin{array}{l} \overset{\bar\alpha^m}{\nearrow} B_1 \\ \underset{\nu}{\searrow} C_1 \end{array},\ \text{with } A_2 \xrightarrow{\alpha^m} B_2 \text{ and } \alpha\in\bar{L_1}\cap L_2,$$

when a similar argument can be applied.

Again we have seen that in both cases the requirement of the confluence definition has been satisfied.

(iv) B is $B_1 \| B_2$, C is $C_1 \| C_2$ and $\mu=\nu=\tau$. Then

$$A_1 \begin{array}{l} \overset{\alpha^m}{\nearrow} B_1 \\ \underset{\beta^n}{\searrow} C_1 \end{array} \quad \text{and} \quad A_2 \begin{array}{l} \overset{\bar\alpha^m}{\nearrow} B_2 \\ \underset{\bar\beta^n}{\searrow} C_2 \end{array}$$

(or likewise with the β and $\bar\beta$ interchanged when the argument is similar but slightly easier as the case $\alpha=\beta$ does not have to be considered). Now if $\alpha=\beta$ then the fact that A_2 is SD_{k+1} tells us that m=n, and hence that $B_1 \sim C_1$ and $B_2 \sim C_2$,giving $B \sim C$. Otherwise we have (since A_1 is SC_{k+1})

$$A_1 \begin{array}{l} \overset{\alpha^m}{\nearrow} B_1 \xrightarrow{\beta^n} D_1 \\ \qquad\qquad\quad \wr \\ \underset{\beta^n}{\searrow} C_1 \xrightarrow{\alpha^m} E_1 \end{array} \quad \text{and} \quad A_2 \begin{array}{l} \overset{\bar\alpha^m}{\nearrow} B_2 \xrightarrow{\bar\beta^n} D_2 \\ \\ \underset{\bar\beta^n}{\searrow} C_2 \xrightarrow{\bar\alpha^m} E_2 \end{array}, \text{ so } \begin{array}{l} B \xrightarrow{\tau} D_1 \| D_2 \\ \qquad\quad \wr \\ C \xrightarrow{\tau} E_1 \| E_2 \end{array}$$

In all these cases we have proved that $A_1 \| A_2$ satisfies the first clause of the definition of SC_{k+1}; we have not, of course, considered all the possible cases, but any others are similar to one of the four considered, and the proofs are symmetrical to those above. To complete the proof of (k+1)-confluence we have to show that if $A_1 \| A_2 \xrightarrow{\mu} B_1 \| B_2$ then $B_1\ B_2$ is SC_k. Now for some μ_1 and μ_2 we have $A_1 \xrightarrow{\mu_1} B_1$ and $A_2 \xrightarrow{\mu_2} B_2$; then as A_1 and A_2 are SCD_{k+1} we can deduce that B_1 and B_2 are SCD_k. Hence by the inductive hypothesis $B_1 \| B_2$ is SCD_k.

This completes the proof that $A_1 \| A_2$ is SC_{k+1} and also proves that the second of the two conditions for determinacy is

satisfied. We now consider the remainder of the proof that $A_1 \| A_2$ is SD_{k+1}. So suppose

$A_1 \| A_2 \xrightarrow{\alpha} B$, $A_1 \| A_2 \xrightarrow{\alpha} C$, then, since $L_1 \cap L_2 = \phi$,

we assume without loss of generality that $\alpha \notin L_2$. Hence it must be the case that

$A_1 \xrightarrow{\alpha} B_1$, $A_1 \xrightarrow{\alpha} C_1$, where B is $B_1 \| A_2$ and C is $C_1 \| A_2$.

Then as A_1 is SD_{k+1} we have $B_1 \sim U\{m/x\}$ and $C_1 \sim U\{n/x\}$ for some U. Then as A_2 has no free variables we can conclude that $B \sim (U \| A_2)\{m/x\}$ and $C \sim (U \| A_2)\{n/x\}$, satisfying the requirements for α-(k+1)-determinacy. A similar argument can be used to show that $A_1 \| A_2$ is $\bar{\alpha}$-SD_{k+1}, completing the proof of (k+1)-determinacy. We have thus shown that the proposition holds for SCD_{k+1}, completing the inductive step.

It should be noted that the determinacy of A_2 was used in the proof of the confluence of A; this helps to explain why it was necessary to introduce determinacy in order to find our strongly confluent subcalculus.

It was shown earlier that + does not preserve strong confluence and determinacy; however, it would be desirable to be able to include some form of ambiguity operation in the confluent determinate subcalculus. For this reason we extend the guarding operation to include composite guards of the form $(\mu_1 | \mu_2 | \mu_3).A$. This behaviour can undergo μ_1-, μ_2- and μ_3-communications in any order to become A. The concept can be defined formally as follows:

Definition

For $n \geq 1$, $(\mu_1|\ldots.|\mu_n)$ is a composite guard whose actions are given as follows.

The behaviour $(\mu_1).A$ has as its only derivation:

$(\mu_1).A \xrightarrow{\mu_1} A$

For $n>1$, the behaviour $(\mu_1|\ldots.|\mu_n).A$ has as its only derivations:

$(\mu_1|\ldots.|\mu_n).A \xrightarrow{\mu_i} (\mu_1|\ldots.|\mu_{i-1}|\mu_{i+1}|\ldots.|\mu_n).A$

for all values of i from 1 to n.

The following strong equivalences can be deduced from this definition of composite guards.

Proposition 3.11

$(\mu_1).A \sim \mu_1.A$

For $n>1$,

$(\mu_1|\ldots.|\mu_n).A \sim \sum_{i=1}^{n} \mu_i.(\mu_1|\ldots.|\mu_{i-1}|\mu_{i+1}|\ldots.|\mu_n).A$

For any permutation p of $(1,\ldots.,n)$,

$(\mu_1|\ldots.|\mu_n).A \sim (\mu_{p(1)}|\ldots.|\mu_{p(n)}).A$

Proof

Immediate from the above definition.

A composite guard such as $(\alpha x|\bar{\beta}x)$ will cause problems if an attempt is made to instantiate its free variables; the x attached to the $\bar{\beta}$ guard could be either free or bound depending on the order of derivations. To avoid such problems such guards will not be allowed.

Definition

A composite guard is well-defined if the intersection of the set of variables accompanying positive labels and the set of those occuring in the expressions accompanying negative labels is empty.

In the following discussions all composite guards will be assumed to be well-defined.

In [Mil4] Milner found that composite guarding preserved strong confluence and determinacy in the pure CCS. However, this is not true here because of problems that occur when the same label appears in two places in a composite guard with different values each time and also when the same variable is attached to more than one label. However, it will be seen that confluence and determinacy are preserved by composite guards that satisfy certain restrictions, which will be introduced after considering examples where confluence and determinacy are not preserved. There will be no intuitive reasons why the constraints chosen should be sufficient but it will be seen that they enable the proof to proceed.

We consider examples of simple behaviours involving composite guards which are not confluent and determinate to motivate the restrictions that will be introduced, and with each example will specify informally the constraint which ensures that it is not allowed in our restricted class of composite guards.

$(\alpha x|\alpha y).\bar{\beta}x.NIL$

This is clearly not determinate as the order in which the αx- and αy-communications occur influences the value bound to x which is passed by the $\bar{\beta}x$-action. To prevent this possibility from occuring we need to impose a restriction that a composite guard does not include more than one α-action for any positive

label $\alpha \in \Delta$.

$(\alpha x | \beta x).\bar{\gamma} x.NIL$

This is not confluent since the value accompanying the $\bar{\gamma}x$-action could be bound to either the x from αx or the x from βx. To prevent this kind of case occuring we shall require that the variables attached to positive labels in a composite guard are all different.

$(\bar{\gamma} 5 | \bar{\gamma} 7).NIL$

This example is not determinate because of the occurence of the negative label $\bar{\gamma}$ twice in the composite guard, attached to different values in each case. The restriction we need to impose to exclude such examples is that if any negative label occurs more than once in a composite guard then in each case the value-expression accompanying it must evaluate to the same value, for any instantiation of values to the free variables of the expressions.

Having considered these examples we are ready to define formally the constraints we wish to impose on composite guards; we shall call a well-defined guard which satisfies these restrictions an admissible composite guard.

Definition

A well-defined composite guard of the form:
$(\alpha_1 x_1 | \ldots . | \alpha_r x_r | \bar{\alpha}_{r+1} e_{r+1} | \ldots . | \bar{\alpha}_s e_s | \tau | \ldots . | \tau)$
(where the x_i are variables and the e_i are value expressions) is said to be admissible if it satisfies the following restrictions:

(i) For $m,n \in \{1,\ldots . r\}$

$$\alpha_m = \alpha_n \Rightarrow m=n$$

(ii) For $m,n \in \{1,\ldots . r\}$

$$x_m = x_n \Rightarrow m=n$$

(iii) For any instantiation of values to the free variables of $e_{r+1},\ldots.e_s$, resulting in values $v_{r+1},\ldots.v_s$ for the expressions then for $m,n\in\{r+1,\ldots.s\}$

$\bar{\alpha}_m=\bar{\alpha}_n \Rightarrow v_m=v_n$

It should be noted that by proposition 3.11 any composite guard can be expressed in the form given at the start of this definition. The second condition above is needed for the preservation of strong confluence; the other two for strong determinacy. It would perhaps be easier to write down tighter constraints which could be expressed more easily (for example insisting that all the α_i, x_i and variables in the e_i are distinct) but this would limit our derived confluent determinate subcalculus unnecessarily.

We are almost ready to prove that admissible composite guards preserve confluence and determinacy; in fact we prove a slightly stronger result which will prove useful when we need to consider recursively-defined behaviours in our derived calculus. However, before proceeding we will need to define confluence and determinacy for general behaviours that may include free variables. This is done in the obvious way in the following definition.

Definition

A behaviour is SC if and only if given any instantiation of values to its free variables the program obtained is SC.
SC_k, SD, SD_k, λ-SD and λ-SD_k are defined likewise for general behaviours.

It is fairly easy to see that the propositions 3.7, 3.8, 3.9 and 3.10 hold for behaviours in general as well as programs, as

these do not involve the introduction of any new free variables, and we will need to use this in showing that programs in the derived calculus are strongly confluent and determinate. However, the following result cannot be extended immediately from programs to behaviours so it has to be formulated in terms of behaviours.

Proposition 3.12

Let $(\mu_1|....|\mu_n)$ be an admissible composite guard. If a behaviour U is SC_k then $(\mu_1|....|\mu_n).U$ is SC_{k+n}. If U is SD_k then $(\mu_1|....|\mu_n).U$ is SD_{k+n}.

Proof

We prove the result for fixed k using induction on n. The confluence result will be proved in detail; a similar technique is used for determinacy.

The result is trivial in the case n=1.

For the inductive step we assume the result holds for n and prove it for n+1. Let A be the program which results from an instantiation of values to the free variables of $(\mu_1|....|\mu_{n+1}).U$. Then we have

$$A = (\mu_1|....|\mu_{n+1}).V,$$

where V is the behaviour obtained by applying the same instantiation to U (note that V need not be a program as some of U's free variables may have been bound from within the composite guard and hence not instantiated). This could not be done if the composite guard was not well-defined in the sense dicussed earlier; this is why that definition was needed. Then V is SC_k (since U is) and we have to show that A is SC_{k+n+1}. Any μ_i-derivative of A is of the form

$$(\mu_1|....|\mu_{i-1}|\mu_{i+1}|....|\mu_{n+1}).V \text{ or}$$

$$(\mu_1|\ldots.|\mu_{i-1}|\mu_{i+1}|\ldots.|\mu_{n+1}).V\{m/x\}$$

and by the inductive hypothesis and proposition 3.8 (generalised to behaviours) we know that this is SC_{k+n}. This has shown that the second clause of the (k+n+1)-confluence definition holds for A and all that is left to prove is that the first clause is satisfied. To do this we assume that A is expressed in the form

$$(\alpha_1 x_1|\ldots.|\alpha_r x_r|\bar{\alpha}_{r+1}v_{r+1}|\ldots.|\bar{\alpha}_s v_s|\tau|\ldots.|\tau).V$$

and consider

$$A \begin{matrix} \nearrow^{\mu} B \\ \searrow_{\nu} C \end{matrix}$$, investigating possible cases for μ and ν.

(i) $\mu=\alpha_i m$ for $i\in\{1,\ldots.r\}$. We may assume without loss of generality that i=1; hence

$$B = (\alpha_2 x_2|\ldots.|\alpha_r x_r|\bar{\alpha}_{r+1}v_{r+1}|\ldots. \ldots.|\bar{\alpha}_s v_s|\tau|\ldots.|\tau).V\{m/x_1\}.$$

Now the possibilities for ν are

<u>either</u> $\nu=\alpha_j n$ for $j\in\{1,\ldots.r\}$, when if j=1 it is easy to see that the last case of the confluence definition is satisfied; otherwise assume without loss of generality that j=r giving

$$C = (\alpha_1 x_1|\ldots.|\alpha_{r-1}x_{r-1}|\bar{\alpha}_{r+1}v_{r+1}|\ldots. \ldots.|\bar{\alpha}_s v_s|\tau|\ldots.|\tau).V\{n/x_r\}.$$

As $\alpha_i \neq \alpha_j$, it follows from the second of the conditions imposed on admissible composite guards that $m \neq n$, so $B \xrightarrow{\nu} D$ and $C \xrightarrow{\mu} D$ where

$$D = (\alpha_2 x_2 | \ldots . | \alpha_{r-1} x_{r-1} | \bar{\alpha}_{r+1} v_{r+1} | \ldots . \\ \ldots . | \bar{\alpha}_s v_s | \tau | \ldots . | \tau) . V(m, n/x_1, x_r),$$

as required;

or $\nu = \bar{\alpha}_j v_j$ for $j \in (r+1, \ldots . s)$; in this case we assume without loss of generality that $j = r+1$. Then

$$C = (\alpha_1 x_1 | \ldots . | \alpha_r x_r | \bar{\alpha}_{r+2} v_{r+2} | \ldots . \\ \ldots . | \bar{\alpha}_s v_s | \tau | \ldots . | \tau) . V,$$

so $B \xrightarrow{\nu} D$ and $C \xrightarrow{\mu} D$ where

$$D = (\alpha_2 x_2 | \ldots . | \alpha_r x_r | \bar{\alpha}_{r+2} v_{r+2} | \ldots . \\ \ldots . | \bar{\alpha}_s v_s | \tau | \ldots . | \tau) . V(m/x_1),$$

as required;

or $\nu = \tau$, when a similar but easier argument can be applied.

(ii) $\mu = \bar{\alpha}_i v_i$ for $i \in (r+1, \ldots . s)$. Assume without loss of generality that $i = r+1$ so

$$B = (\alpha_1 x_1 | \ldots . | \alpha_r x_r | \bar{\alpha}_{r+2} v_{r+2} | \ldots . \\ \ldots . | \bar{\alpha}_s v_s | \tau | \ldots . | \tau) . V.$$

Now considering the possibilities for ν,

either $\nu = \alpha_j m$ for $j \in (1, \ldots . r)$, which is the same as the second case investigated for (i) with the μ- and ν-derivations interchanged;

or $\nu = \bar{\alpha}_j v_j$ for $j \in (r+1, \ldots . s)$, when if $i = j$ we have $\mu = \nu$ and $B = C$; otherwise we may assume without loss of generality that $i = s$ so

$$C = (\alpha_1 x_1 | \ldots . | \alpha_r x_r | \bar{\alpha}_{r+1} v_{r+1} | \ldots . \\ \ldots . | \bar{\alpha}_{s-1} v_{s-1} | \tau | \ldots . | \tau) . V,$$

and $B \xrightarrow{\nu} D$ and $C \xrightarrow{\mu} D$ where

$$D = (\alpha_1 x_1 | \ldots . | \alpha_r x_r | \bar{\alpha}_{r+2} v_{r+2} | \ldots . \ldots . | \bar{\alpha}_{s-1} v_{s-1} | \tau | \ldots . | \tau) . V,$$

as required;

or $\nu = \tau$, when the argument used is similar but more straightforward.

(iii) $\mu = \tau$, when if $\nu = \tau$ then $B \equiv C$, else we have one of the cases already considered with the roles of μ and ν reversed.

In all cases it has been demonstrated that the requirements of the strong confluence definition have been satisfied, hence A is SC_{k+n+1} as required to complete the inductive step.
The result for determinacy is proved in a similar manner, making use in appropriate cases of the restrictions (i) and (iii) from the definition of admissible composite guards.

Having proved this result we can state that all finite programs in the derived algebra built from NIL, restriction, relabelling, restricted disjoint composition and admissible composite guards are strongly confluent and determinate, but we need to consider recursively-defined behaviours before it is possible to deduce that the subcalculus is confluent and determinate. The increase in level from k-confluence and determinacy to k+n in the previous result will enable us to handle recursion; however an increase of 1 is adequate so it is convenient to work with the following corollary rather than proposition 3.12 itself.

Corollary 3.13

If a behaviour U is SC_k (or SD_k) and $(\mu_1 | \ldots . | \mu_n)$ is an admissible composite guard then $(\mu_1 | \ldots . | \mu_n) . U$ is SC_{k+1} (or SD_{k+1} respectively).

Proof

Follows immediately from proposition 3.12.

It is desirable to include conditional expressions in the derived subcalculus, hence the following result is required:

Proposition 3.14

If the behaviours U_1 and U_2 are both SCD then, for any boolean expression E, the behaviour **if** E **then** U_1 **else** U_2 is SCD.

Proof

We have to show that any program obtained by the instantiation of values to the free variables of **if** E **then** U_1 **else** U_2 is SCD. Such a program must take the form of either **if** true **then** B_1 **else** B_2 or **if** false **then** B_1 **else** B_2, and is hence strongly congruent to either B_1 or B_2. By propositions 3.2 and 3.6 it is thus sufficient to show that B_1 and B_2 are SCD, and since these programs were obtained by an instantiation to free variables of U_1 and U_2 the result follows immediately from the assumption that U_1 and U_2 are SCD.

Recalling that recursively-defined behaviour identifiers are allowed in CCS only if they are guardedly well-defined (as discussed in the previous chapter) we are now able to state and prove the result that all behaviours in the derived subcalculus are strongly confluent and determinate.

Definition

The derived subcalculus CCCS (confluent CCS) comprises the operations inaction (NIL), rd-composition, admissible composite guarding, restriction and relabelling, along with conditional expressions.

Proposition 3.15

Every behaviour identifier in CCCS is SCD_k for all k.

Proof

By induction on k. The result is trivially true for k=0, so we prove the inductive step at k+1. Suppose we have a behaviour identifier V defined by $V \Leftarrow B_V$. By guarded well-definedness this may be expanded (by substituting B_b for any behaviour identifier b) until every behaviour identifier is guarded. So we have $V \Leftarrow B'_V$ containing no unguarded behaviour identifier. By the inductive hypothesis we know that every behaviour in B'_V is SCD_k, so from propositions 3.7, 3.8, 3.9, 3.10, corollary 3.13 and proposition 3.14 we can deduce that B'_V is SCD_{k+1}, the raising from k to k+1 coming from corollary 3.13 since every behaviour identifier is guarded. Now $V \sim B'_V$ so from propositions 3.2 and 3.6 we can deduce that V is SCD_{k+1} as required.

We have proved that every behaviour in CCCS is strongly confluent and hence it is possible to apply the Strong Confluence Theorem freely to programs in this derived calculus. We are ready to introduce an example to show how this can be useful in proving that systems meet their specifications.

AN EXAMPLE - PROVING THAT A BUFFER MEETS ITS SPECIFICATION

As a first example of the application of confluence techniques to proofs about behaviours involving value-passing we will consider a construction of a bounded buffer built in CCCS, and show how the Strong Confluence Theorem can be used in the proof that the buffer meets its specification.

We first give an informal specification of the buffer; it will hold a list of at most n elements from a finite set A and will be capable of accepting values by an ι-communication and placing them at the head of the list provided its length is less than n. Whenever the list is non-null the buffer will be capable of removing the tail element and passing it in an o-communication. The specification can be expressed more formally as follows:

$$B(\varepsilon) = \iota y.B(y:\varepsilon);$$
$$B(s:x) = \iota y.B(y:s:x) + o x.B(s), \quad \text{for } |s| < n-1; \text{ and}$$
$$B(s:x) = o x.B(s) \quad \text{for } |s| = n-1;$$

where $x,y \in A$, $s \in A^*$ and ':' represents concatenation of elements to strings at either end.

The buffer will be constructed as a chain of n cells which can hold an element of A or be empty. For convenience of notation we will assume that 0 is not an element of A and represent an empty cell by CELL(0). A cell will be defined in the following way:

$$CELL(x) = \underline{\text{if}}\ x=0\ \underline{\text{then}}\ \alpha y.CELL(y)\ \underline{\text{else}}\ \bar{\beta} x.CELL(0).$$

To be able to identify a cell's position in the buffer and ensure that its value-exchanges take place with adjacent cells we define

$$CELL_i(x) = CELL(x)[\alpha_{i-1}, \alpha_i / \alpha, \beta]$$

for i from 1 to n. We then construct a chain from the n cells by using rd-composition to produce a behaviour $(CELL_1(x_1)\|....\|CELL_n(x_n))$ of sort $\{\alpha_0,\bar{\alpha}_n\}$. (It was asserted in [Mil4] that rd-composition is associative; this result also holds in the value-passing case so there is no ambiguity in the expression). To obtain the buffer to meet the specification it is necessary to relabel the α_0 and α_n to ι and σ, so we have

$$BUFF(x_1,....,x_n) = (CELL_1(x_1)\|....\|CELL_n(x_n))[\iota,\sigma/\alpha_0,\alpha_n].$$

In order to obtain the lists as in the specification from the n-tuples that parameterise BUFF we define

$$\begin{aligned} B(\varepsilon) &= BUFF(0,....,0); \\ B(s) &= BUFF(0,....,0,x_1,....,x_k) \\ &\qquad \text{when } s = \langle x_1,....,x_k\rangle \text{ for } k<n; \text{ and} \\ B(s) &= BUFF(x_1,....,x_n) \\ &\qquad \text{when } s = \langle x_1,....,x_n\rangle. \end{aligned}$$

The aim now is to prove that the parameterised behaviour B just defined does satisfy the specifications above. We begin by using the expansion theorem (2.8) to obtain the following strong equivalence:

$$\begin{aligned} BUFF(x_1,....,x_n) \sim\ & \underline{\text{if}}\ x_1=0\ \underline{\text{then}}\ \iota y.BUFF(y,x_2,....,x_n) \\ & \qquad \underline{\text{else}}\ NIL \\ & + \underline{\text{if}}\ x_n\neq 0\ \underline{\text{then}}\ \bar{\sigma}x_n.BUFF(x_1,....,x_{n-1},0) \\ & \qquad \underline{\text{else}}\ NIL \\ & + \sum\{\tau.BUFF(x_1,....,x_{i-1},0,a,x_{i+2},....x_n): \\ & \qquad x_i=a\neq 0,\ x_{i+1}=0,\ 0<i<n\}. \end{aligned}$$

From this we can conclude that for $0<i<n$ and $a\in A$

$$BUFF(x_1,....,x_{i-1},a,0,x_{i+2},....,x_n) \xrightarrow{\tau} BUFF(x_1,....,x_{i-1},0,a,x_{i+2},....,x_n)$$

Since $BUFF(x_1,\ldots,x_n)$ was defined using the behaviour constructs of CCCS it follows from proposition 3.15 that it is SCD; hence we can apply the Strong Confluence Theorem giving the result

$$BUFF(x_1,\ldots,x_{i-1},a,0,x_{i+2},\ldots,x_n) \approx BUFF(x_1,\ldots,x_{i-1},0,a,x_{i+2},\ldots,x_n).$$

Having obtained the expansion above and deduced an equivalence of chains carrying the same strings but with "gaps" in different places, we are now ready to prove that the constructed buffer meets its specification. The first thing we have to show is that

$$B(\varepsilon) = \iota y.B(y:\varepsilon).$$

From the definition we have

$$\begin{aligned} B(\varepsilon) &= BUFF(0,\ldots,0) \\ &= \iota y.BUFF(y,0,\ldots,0), \end{aligned}$$

using the expansion result above. Then applying several times the equivalence that was obtained using the Strong Confluence Theorem we conclude that

$$BUFF(y,0,\ldots,0) \approx BUFF(0,\ldots,0,y)$$

and as guarding turns observational equivalence into congruence it follows that

$$\iota y.BUFF(y,0,\ldots,0) = \iota y.BUFF(0,\ldots,0,y)$$

so we can conclude

$$\begin{aligned} B(\varepsilon) &= \iota y.BUFF(0,\ldots,0,y) \\ &= \iota y.B(y:\varepsilon) \end{aligned}$$

as required.

The next part of the proof that the buffer meets its specification is to show that

$$B(s{:}x) = \iota y.B(y{:}s{:}x) + \bar{\sigma}x.B(s)$$

for strings s of length less than n-1; so let $s=\langle x_1,\ldots,x_{k-1}\rangle$ for $0<k<n$. Then we have

$$\begin{aligned} B(s{:}x) &= BUFF(0,\ldots,0,x_1,\ldots,x_{k-1},x) \\ &= \iota y.BUFF(y,0,\ldots,0,x_1,\ldots,x_{k-1},x) \\ &\qquad + \bar{\sigma}x.BUFF(0,\ldots,0,x_1,\ldots,x_k-1,0) \end{aligned}$$

using the expansion result. Then by several applications of the equivalence obtained from the Strong Confluence Theorem we obtain

$$\begin{aligned} &BUFF(y,0,\ldots,0,x_1,\ldots,x_{k-1},x) \\ &\qquad \approx BUFF(0,\ldots,0,y,x_1,\ldots,x_{k-1},x) \text{ and} \\ &BUFF(0,\ldots,0,x_1,\ldots,x_{k-1},0) \\ &\qquad \approx BUFF(0,\ldots,0,x_1,\ldots,x_{k-1}). \end{aligned}$$

From these we obtain the congruence

$$\begin{aligned} &\iota y.BUFF(y,0,\ldots,0,x_1,\ldots,x_{k-1},x) \\ &\qquad + \bar{\sigma}x.BUFF(0,\ldots,0,x_1,\ldots,x_{k-1},0) \\ &= \iota y.BUFF(0,\ldots,0,y,x_1,\ldots,x_{k-1},x) \\ &\qquad + \bar{\sigma}x.BUFF(0,\ldots,0,x_1,\ldots,x_{k-1}) \end{aligned}$$

so we can conclude that

$$B(s{:}x) = \iota y.B(y{:}s{:}x) + \bar{\sigma}x.B(s)$$

as required.

Finally, we have to show that

$$B(s{:}x) = \bar{\sigma}x.B(s)$$

for strings s of length n-1; let $s=\langle x_1,\ldots,x_{n-1}\rangle$, then

$$B(s:x) = BUFF(x_1,\ldots,x_{n-1},x)$$
$$= \bar{\sigma}x.BUFF(x_1,\ldots,x_{n-1},0)$$

using the expansion result. Then we apply the equivalence deduced from the Strong Confluence Theorem n-1 times to obtaain

$$BUFF(x_1,\ldots,x_{n-1},0) \simeq BUFF(0,x_1,\ldots,x_{n-1})$$

and hence we have

$$\bar{\sigma}x.BUFF(x_1,\ldots,x_{n-1},0) = \bar{\sigma}x.BUFF(0,x_1,\ldots,x_{n-1})$$

which gives us the required result

$$B(s:x) = \bar{\sigma}x.B(s).$$

The proof that the constructed buffer meets its specification has now been completed; note that the Strong Confluence Theorem was used in all three parts of the proof (although we did not apply it directly but instead first investigated τ-actions that occured in the constructed behaviour and formulated what was essentially an equivalence schema as an application of the theorem). Without the results on confluence this example would have involved much more work in order to prove that the specification was satisfied.

OBSERVATION CONFLUENCE AND DETERMINACY

The definitions of confluence and determinacy in this chapter have involved only single-length derivations ($\xrightarrow{\mu}$) and, accordingly, the properties were preserved only by strong equivalence, $\sim$, and not observational equivalence; hence the names strong confluence and strong determinacy. In [Mil4] Milner

showed how to define a weaker observational confluence and determinacy for the pure CCS, which were preserved by $\approx$. His definitions were quite lengthy even without the extra detail needed for value-passing so it is not felt appropriate to extend them to the full CCS here; however, in a subsequent chapter an alternative definition of observational equivalence will be introduced for which it has been found that it is irrelevant whether single- or multiple-length derivations are considered; hence we will investigate the role of observation confluence in this new setting later.

4 UNIQUENESS OF SOLUTION OF RECURSIVE BEHAVIOUR EQUATIONS

INTRODUCTION

In this chapter we consider behaviour equations of the form $b \simeq F[b]$ and investigate what constraints need to be imposed on F to ensure that the solution is unique up to observational equivalence. It is not difficult to find simple classes of functionals, F, whose fixed points are unique; for example it was left as an exercise in [Mil4] to show that if $B \simeq^C \lambda.B$ and $C \simeq^C \lambda.C$ then $B \simeq^C C$ - the proof of this will be given below.

It is easy to show that there are non-trivial functionals F which give non-unique solutions to the equation $b \simeq F[b]$. Consider, for example the equation

$$b \simeq b + \alpha.NIL,$$

which is satisfied by any behaviour of the form $A + \alpha.NIL$. The functional in this case does not, of course, satisfy the constraint of guarded well-definedness imposed by Milner, and outlined in chapter 2 here; hence it could not be used as a behaviour-defining functional. However, it can be demonstrated that guarded well-definedness does not by itself imply uniqueness of fixed points; consider the equation

$$b \simeq \alpha.(b|\bar{\alpha}.NIL)\backslash\alpha,$$

and let A be any agent whose sort L is such that $\alpha \notin L \cup \bar{L}$. Then

$$\begin{aligned} \alpha.(\alpha.A|\bar{\alpha}.NIL)\backslash\alpha &= \alpha.\tau.(A|NIL)\backslash\alpha \\ &= \alpha.A\backslash\alpha \\ &= \alpha.A \end{aligned}$$

(by the restriction on the sort of A), so $\alpha.A$ satisfies the

equation. Many agents A satisfy the condition imposed on the sort; hence the equation $b \simeq \alpha.(b|\bar{\alpha}.NIL)\backslash\alpha$ has many solutions.

The examples considered above have demonstrated that functionals (or contexts) do not have unique fixed points in certain cases where unguarded ambiguity occurs and where certain combinations of composition and restriction lead to the "swallowing" of guards (as with the α in the last example).

CONTRACTING TRANSFORMATIONS

A recursive behaviour definition such as

$$b(x) \Leftarrow \bar{\alpha}x.NIL + \beta y.b(x+y)$$

defines b to satisfy

$$b \simeq^{C} (x).(\bar{\alpha}x.NIL + \beta y.b(x+y));$$

that is, b is defined as a fixed point (up to $\simeq^{C}$) of the behaviour functional F given by

$$F[b] \equiv (x).(\bar{\alpha}x.NIL + \beta y.b(x+y)).$$

The notation (x).E means that the expression E is a function of x and is similar to abstraction in the λ-calculus [HLS].

Such behaviour functionals will usually be referred to as transformations. We are aiming to find classes of transformations which have unique fixed points up to $\simeq$; such transformations will be said to be contracting, this term being inspired by the concept of contraction mappings in metric spaces [Sut], which will be discussed at the end of this chapter.

Definition

A contracting transformation F is one which has the property that whenever $b_1 \approx F[b_1]$ and $b_2 \approx F[b_2]$ then $b_1 \approx b_2$.

If a transformation F is contracting and a behaviour b is defined by $b \Leftarrow F[b]$ (which specifies b as a particular fixed point of F) and we can prove that $b' \approx F[b']$, then we can conclude $b \approx b'$. Furthermore, if it is known that b and b' are stable then it can be concluded that $b \approx^C b'$.

Returning to the buffer example of the previous chapter, recall the specification was in the form:

$B(\varepsilon) = \iota y.B(y:\varepsilon)$
$B(s:x) = \ldots.$

This can be expressed as B = F[B] where

$F[B] \equiv (s).$ **if** $s=\varepsilon$ **then** $\iota y.B(y:\varepsilon)$
else ....

Let B_0 be the specification buffer, defined by $B_0 \Leftarrow F[B_0]$, and let B be the buffer constructed in the previous chapter. Then $B_0 \approx^C F[B_0]$ and $B \approx^C F[B]$, so, assuming F is contracting (and it is believed that it is - although parameterised transformations are not considered in the investigation of contracting transformations that follows), then we have $B \approx B_0$. Hence B is the unique solution of the specification up to observational equivalence. Furthermore, as $F[B] \approx F[B_0]$ and F[B] and $F[B_0]$ are both stable it follows that $F[B] \approx^C F[B_0]$; hence $B \approx^C B_0$ and the constructed buffer is the unique solution up to observational congruence.

In our search for contracting transformations we begin by showing that $b \approx \alpha.b$ has a unique solution up to observational

equivalence. This result was set as an exercise (7.7) in [Mil14].

Proposition 4.1

The transformation F defined by $F[b] \equiv \alpha.b$ is contracting.

Proof

Assuming $b_1 \approx \alpha.b_1$ and $b_2 \approx \alpha.b_2$, we have to prove that $b_1 \approx b_2$. This is done using induction on k to show that $b_1 \approx_k b_2$ for all $k \geq 0$. The result is trivial for k=0; we consider the inductive step at k+1. For this we assume $b_1 \stackrel{s}{\Rightarrow} c_1$ and have to show that $b_2 \stackrel{s}{\Rightarrow} c_2$ for some c_2 such that $c_1 \approx_k c_2$. To do this a second induction is used, on the length, n, of the string s. The two cases to consider are:

(i) The base step n=0, with $s=\varepsilon$. We have $b_1 \stackrel{\varepsilon}{\Rightarrow} c_1$ and $b_1 \approx \alpha.b_1$; hence $\alpha.b_1 \stackrel{\varepsilon}{\Rightarrow} d_1$ with $d_1 \approx_k c_1$. As $\alpha.b_1$ is stable it follows that $d_1 \equiv \alpha.b_1$, so $c_1 \approx_k \alpha.b_1$. Using the inductive hypothesis (for the k-induction) $b_2 \approx_k b_1 \approx \alpha.b_1 \approx_k c_1$, so setting c_2 equal to b_2 we have $b_2 \stackrel{\varepsilon}{\Rightarrow} c_2$ and $c_1 \approx_k c_2$ as required; and

(ii) The inductive step with $|s| = n+1$, where we can assume $s=\lambda s'$. As $b_1 \stackrel{\lambda}{\Rightarrow} c_1$ and $b_1 \approx \alpha.b_1$ we have $\alpha.b_1 \stackrel{\lambda}{\Rightarrow} d_1$ for some $d_1 \approx_k c_1$. This means that $\alpha.b_1 \stackrel{\lambda s'}{\Rightarrow} d_1$, so $\alpha=\lambda$ and $\alpha.b_1 \stackrel{\alpha}{\Rightarrow} b_1 \stackrel{s'}{\Rightarrow} d_1$. Then as $|s'|=n$, the inductive hypothesis gives $b_2 \stackrel{s'}{\Rightarrow} d_2$ with $d_2 \approx_k d_1$, hence $d_2 \approx_k c_1$. So $\alpha.b_2 \stackrel{s}{\Rightarrow} d_2$, and as $b_2 \approx \alpha.b_2$ it follows that $b_2 \stackrel{s}{\Rightarrow} c_2$ for some c_2 such that $c_2 \approx_k d_2 \approx_k c_1$, as required.

The same argument can be used to show that whenever $b_2 \stackrel{s}{\Rightarrow} c_2$ then $b_1 \stackrel{s}{\Rightarrow} c_1$ with $c_1 \approx_k c_2$. Hence $b_1 \approx_{k+1} b_2$, completing the inductive step of the proof.

The α in the above proposition could be replaced by any $\lambda \in \Lambda$ (or any $\mu \in \Lambda \times V$ in the value-passing case) without any alteration to the method of proof. Additionally, a similar argument can be used to demonstrate that any transformation F of the form $F[b] \equiv \mu_1.\ldots.\mu_n.b$ with at least one of the μ_i not equal to τ is contracting and also when $F_1,\ldots,F_n$ are all of this form then the transformation G defined by

$$G[b] \equiv F_1[b] + \ldots + F_n[b]$$

is contracting. However, it is not true in general that when F and G are contracting transformations then so is the transformation H defined by

$$H[b] \equiv F[b] + G[b].$$

To demonstrate this we consider an example similar to that introduced earlier where a guard was "swallowed"; here mutual "swallowing" occurs. Let F, G and H be defined by

$$F[b] \equiv \alpha.(b|\bar{\beta}.NIL)\backslash L$$
$$G[b] \equiv \beta.(b|\bar{\alpha}.NIL)\backslash L,$$
$$\text{and } H[b] \equiv F[b] + G[b]$$

where $L = \{\alpha,\beta\}$. We will show that, although F and G are contracting, the equation $b \approx H[b]$ has many solutions. Firstly, to demonstrate that F is contracting let b be a solution of the equation $b \approx F[b]$, having sort L'. Then (observing that $\beta \notin L'$)

$$\begin{aligned} b &\approx \alpha.(b|\bar{\beta}.NIL)\backslash L \\ &= \alpha.(b\backslash L|(\bar{\beta}.NIL)\backslash L) \quad (\text{as } \alpha,\beta \notin \text{names}(L' \cap \{\bar{\beta}\})) \\ &= \alpha.(b\backslash L|NIL) \\ &= \alpha.b\backslash L. \end{aligned}$$

Then, as observational equivalence is preserved by restriction,

$$b\backslash L \approx (\alpha.b\backslash L)\backslash L$$
$$= NIL.$$

This gives us (as guarding preserves observational equivalence)

$$\alpha.b\backslash L \approx \alpha.NIL,$$

and we have seen above that $b \approx \alpha.b\backslash L$, so

$$b \approx \alpha.NIL.$$

This holds for any b satisfying $b \approx F[b]$, and hence F is a contracting transformation. The same argument with α and β interchanged shows that G is also contracting. To show that the equation $b \approx H[b]$ does not have a unique solution up to $\approx$ the argument used is similar to the one applied to the example in the introduction. We demonstrate that for any agent A of sort L' such that $names(L') \cap L = \emptyset$, the behaviour B defined as $\alpha.A + \beta.A$ satisfies $B \approx H[B]$:

$$\begin{aligned} H[B] &= F[B] + G[B] \\ &= \alpha.(B|\bar{\beta}.NIL)\backslash L + \beta.(B|\bar{\alpha}.NIL)\backslash L \\ &= \alpha.(\tau.(A|NIL)\backslash L) + \beta.(\tau.(A|NIL)\backslash L) \\ &= \alpha.A + \beta.A \text{ (as } names(L') \cap L = \emptyset) \\ &= B. \end{aligned}$$

This completes the proof that B satisfies the equation $B \approx H[B]$, and since this holds for any B of the form $\alpha.A + \beta.A$ (with the appropriate restriction on A's sort) it is concluded that H is not a contracting transformation.

DERIVATIONS OF CONTEXTS

In order to introduce a class of contracting transformations we first need to investigate how the possible derivations of F[b]

depend on the possible derivations of b. This study will be restricted to simple contexts F of the form $F[b] \equiv (....b....)$, in which the expression defining F[b] contains only a single occurrence of b and no other behaviour identifiers. For convenience the definitions will be introduced in the pure CCS without value-passing; it is not anticipated that there should be any difficulty in extending the concepts to the full calculus, but no attempt has been made to do so.

The basic concept to be introduced is that of an action of a behaviour "causing" an action of that behaviour within a context (the idea being suggested by Robin Milner). For example, if $F[b] \equiv b[\beta/\alpha]$ and $A \xrightarrow{\alpha} A'$ then the α-action of A can be said to "cause" the derivation $F[A] \xrightarrow{\beta} F[A']$. This is valid for any A which can undergo the α-action, hence it is reasonable to say that F and the α-action cause a β-action resulting in F. Considering another example let $G[b] \equiv (b|\bar{\alpha}.NIL)\backslash\alpha$. Then if $A \xrightarrow{\alpha} A'$ we have $G[A] \xrightarrow{\tau} G'[A']$, where $G'[b] \equiv (b|NIL)\backslash\alpha$. Here G and the α-action have caused a τ-action resulting in G'. We are ready to define this concept formally:

Definition

For $\mu, \nu \in \Lambda \cup \{\tau\}$ and contexts F and F' it is said that F, μ causes ν, F' (written $F[\ ] \xrightarrow{\nu}_{\mu} F'[\ ]$) if and only if

$$x \xrightarrow{\mu} y \vdash F[x] \xrightarrow{\nu} F'[y].$$

The $\vdash$ notation means simply that given a proof that $x \xrightarrow{\mu} y$ it is possible to construct a proof that $F[x] \xrightarrow{\nu} F'[y]$ - a proof is understood to be a sequence of steps obtained from the operational semantics schema.

It is not always the case that derivations of contexts are caused in the manner of the above definition, for example let $F[b] \equiv a.b$, then $F[A] \xrightarrow{a} F'[A]$ where F' is the identity context, for any agent A - even one which has no derivations. In this case, however, the derivation of F[A] was independent of A, and it is true that all derivations of any behaviour of the form F[A] are either independent of A or "caused" by F in the sense of the definition. This is proved in the lemma that follows; however we first introduce notation to encapsulate the fact that a derivation of F[A] is independent of A.

Definition

For $\mu \in \Lambda \cup \{\tau\}$ and contexts F and F', the notation $F[\ ] \xrightarrow{\mu} F'[\ ]$ means that $\vdash F[A] \xrightarrow{\mu} F'[A]$ for any behaviour A. (That is, a proof of $F[A] \xrightarrow{\mu} F'[A]$ can be obtained without making any assumptions about A.)
$F[\ ] \xRightarrow{s} F'[\ ]$ is defined analogously for $s \in \Lambda^*$.

Lemma 4.2

If $F[A] \xrightarrow{\nu} D$ for $\nu \in \Lambda \cup \{\tau\}$ then either

(i) $F[\ ] \xrightarrow{\nu}_{\mu} F'[\ ]$ and $A \xrightarrow{\mu} B$ for some $\mu \in \Lambda \cup \{\tau\}$, behaviour B and context F' such that D is F'[B]; or

(ii) $F[\ ] \xrightarrow{\nu} F'[\ ]$ for some F', and D is F'[A].

Proof

By structural induction on F. There are seven cases to consider:

(a) $F[b] \equiv C$
(b) $F[b] \equiv b$
(c) $F[b] \equiv \mu.G[b]$
(d) $F[b] \equiv G[b]\backslash\alpha$
(e) $F[b] \equiv G[b]+C$
(f) $F[b] \equiv G[b]|C$
(g) $F[b] \equiv G[b][S]$

(C is an expression containing no behaviour identifiers)

(Any other possibility can be regarded as one of these cases by symmetry; as explained in chapter 2 conditional expressions are omitted from the pure calculus.) For cases (a), (b) and (c) it is found that the result can be proved directly; the inductive hypothesis is needed in the other four cases. We investigate here cases (a), (b) and (c) and also (f) as typical of the use of the induction; the result can be proved for the remaining cases by similar arguments.

(a) $F[A] \equiv C$.
Here the second part of the lemma holds trivially, defining $F'[b] \equiv D$.

(b) $F[A] \equiv A$, so $A \xrightarrow{\nu} D$.
Defining F' to be equal to F, $B \equiv D$ and $\mu=\nu$, it is easy to see that the first clause of the lemma holds, since $F[\ ] \xrightarrow[\nu]{\nu} F[\ ]$ (as a trivial consequence of the definition).

(c) $F[A] \equiv \aleph.G[A]$ so $\aleph.G[A] \xrightarrow{\nu} D$.
It follows that $\nu=\delta$ and $D \equiv G[A]$, hence with F' defined to be $G[A]$ the second clause of the lemma holds.

(f) $F[A] \equiv G[A]|C$, so $G[A]|C \xrightarrow{\nu} D$.
This derivation can occur in three ways,
<u>either</u> $G[A] \xrightarrow{\nu} K$ with $D \equiv K|C$, in which case by the structural induction we have either
(i) $G[\ ] \xrightarrow[\mu]{\nu} G'[\ ]$ and $A \xrightarrow{\mu} B$ with $K \equiv G'[B]$ in which case setting $F'[b] \equiv G'[b]|C$ we have $F[\ ] \xrightarrow[\mu]{\nu} F'[\ ]$, and $D \equiv K|C \equiv G'[B]|C \equiv F'[B]$ as required; or
(ii) $G[\ ] \xrightarrow{\nu} G'[\ ]$ with $K \equiv G'[A]$, when setting $F'[b] \equiv G'[b]|C$ it follows that $F[\ ] \xrightarrow{\nu} F'[\ ]$ and $D \equiv K|C \equiv G'[A]|C \equiv F'[A]$ as required;
<u>or</u> $C \xrightarrow{\nu} K$ with $D \equiv G[A]|K$. In this case setting $F'[b] \equiv G[b]|K$ we have $F[\ ] \xrightarrow{\nu} F'[\ ]$ and $D \equiv F[A]$, so

the second clause of the lemma is satisfied;

or $\nu=\tau$ and $G[A] \xrightarrow{\lambda} J$ and $C \xrightarrow{\Gamma} K$ for some $\lambda \in \Lambda$, and $D \equiv J|K$. Here using the structural induction hypothesis we have either

(i) $G[\ ] \xrightarrow{\lambda} G'[\ ]$ and $A \xrightarrow{\mu} B$ with $J \equiv G'[B]$; then defining $F'[b] \equiv G'[b]|K$, it follows that $F[\ ] \xrightarrow{\tau} F'[\ ]$ and $D \equiv J|K \equiv G'[B]|K \equiv F'[B]$ as required; or

(ii) $G[\ ] \xrightarrow{\lambda} G'[\ ]$ with $J \equiv G'[A]$, in which case, defining $F'[b] \equiv G'[b]|K$ we have $F[\ ] \xrightarrow{\tau} F'[\ ]$ and $D \equiv J|K \equiv G'[A]|K \equiv F'[A]$ as required.

All the possible derivations $G[A]|C \xrightarrow{\nu} D$ have been considered, completing the proof that the result holds in case (f).

Cases (d), (e) and (g) can be treated with similar arguments to the above to complete the proof of the lemma.

We now present three results involving $F[\ ] \xrightarrow{\nu} G[\ ]$ that will be needed later when investigating the effects of unobservable actions when we introduce the property of $F[\ ] \overset{t}{\Rightarrow} F'[\ ]$ for multiple-length derivations.

Proposition 4.3

If $F[\ ] \xrightarrow{\nu} G[\ ]$ then either $\nu=\tau$ or $F[\ ] \xrightarrow{\nu} G[\ ]$.

Proof

By structural induction on F. There are seven cases to consider as listed in the proof of lemma 4.2:

(a) $F[b] \equiv C$.

Here whenever $A \xrightarrow{\tau} B$ then $C \xrightarrow{\nu} G[B]$, so G must be

constant, say $G[b] \equiv D$. Then as $C \xrightarrow{\nu} D$ we have $F[\ ] \xrightarrow{\nu} G[\ ]$.

(b) $F[b] \equiv b$.
In this case whenever $A \xrightarrow{\tau} B$ then $A \xrightarrow{\nu} G[B]$, so it must follow that $\nu = \tau$.

(c) $F[b] \equiv \gamma.F'[b]$.
Here whenever $A \xrightarrow{\tau} B$, $\gamma.A \xrightarrow{\nu} G[B]$, hence $\nu = \gamma$ and $G[B] \equiv F'[A]$. It must be the case that G and F' are constant and $G \equiv F'$, hence $F[\ ] \xrightarrow{\nu} G[\ ]$.

(d) $F[b] \equiv F'[b]\backslash\alpha$.
In this case it can be deduced that $F'[\ ] \xrightarrow{\nu} G'[\ ]$ for some G' such that $G[b] \equiv G'[b]\backslash\alpha$ and $\nu \neq \alpha$, hence by the structural-inductive hypothesis either $\nu = \tau$ or $F'[\ ] \xrightarrow{\nu} G'[\ ]$, which implies that $F[\ ] \xrightarrow{\nu} G[\ ]$.

(e) $F[b] \equiv F'[b] + C$.
In this case either $F'[\ ] \xrightarrow{\nu} G[\ ]$, in which case applying the inductive hypothesis either $\nu = \tau$ or $F'[\ ] \xrightarrow{\nu} G[\ ]$, which implies that $F[\ ] \xrightarrow{\nu} G[\ ]$, or $C \xrightarrow{\nu} G[\ ]$, in which case the result follows as in (a).

(f) $F[b] \equiv F'[b] \mid C$.
Considering the ways that $F[\ ] \xrightarrow{\nu} G[\ ]$ can occur, as in the previous proof, and applying the inductive hypothesis as appropriate gives the desired result in this case.

(g) $F[b] \equiv F'[b][S]$.
The proof in this case is similar to that in case (d).

The result has been proved in all seven cases, completing the structural induction.

Proposition 4.4

If $F[\ ] \xrightarrow{\tau} G[\ ]$ and $G[\ ] \xrightarrow{\mu} G'[\ ]$ then either $F[\ ] \xrightarrow{\tau} G[\ ]$ or $F[\ ] \xrightarrow{\mu} G'[\ ]$.

Proof

This proof is again done by structural induction on F, considering the seven possible cases as listed previously. In case (a) the first result holds trivially and in case (b) it can be shown that $F \equiv G$ so the second result holds. In case (c) it can be shown that $\mu=\nu$ and G is constant and equal to F', hence the first result holds. The structural inductive hypothesis produces the result easily in cases (d) and (g). In case (e) $F[b] \equiv F'[b]+C$, so either $F'[\ ] \xrightarrow{\tau} G[\ ]$, in which case the result is obtained from the inductive hypothesis, or $C \xrightarrow{\tau} G[\ ]$ in which case G is constant and $F[\ ] \xrightarrow{\tau} G[\ ]$. In case (f) $F[b] \equiv F'[b]|C$, and as usual in this case there are three ways the derivation can occur, and the result can be proved easily in each case, using the inductive hypothesis when necessary.

Before proving the third result referred to above we introduce a new piece of notation that will be used regularly later in this chapter and is useful in expressing the next proposition; as much subsequent work will involve strings in Λ^*, $(\Lambda \cup \{\tau\})^*$ and $\Lambda^+ \cup \{\tau\}$ and it will often be necessary to translate between them we define a function $\hat{}$ which maps strings in $(\Lambda \cup \{\tau\})^*$ to Λ^* by removing the τ's.

Definition

The function $\hat{}: (\Lambda \cup \{\tau\})^* \rightarrow \Lambda^*$ is defined as follows:

$\hat{\varepsilon} = \varepsilon$

$\widehat{\tau\sigma} = \hat{\sigma}$

$\widehat{\lambda\sigma} = \lambda\hat{\sigma}$ for $\lambda \in \Lambda$

Having introduced this notation we shall use it in the simple case of $\hat{\nu}$ for $\nu \in \Lambda \cup \{\tau\}$ in expressing the remaining result to be proved about $F[\ ] \xrightarrow[\tau]{\nu} G[\ ]$.

Proposition 4.5

If $F[\ ] \xrightarrow[\tau]{\tau} G[\ ]$ and $G[\ ] \xrightarrow{\nu} G'[\ ]$ for $\nu \in \Lambda \cup \{\tau\}$ then $F[\ ] \overset{\hat{\nu}}{\Rightarrow} G'[\ ]$.

Proof

A proof similar to that for proposition 4.4 above is used to show that either $F[\ ] \xrightarrow{\tau} G[\ ]$, or $F[\ ] \xrightarrow{\nu} G'[\ ]$. In both cases it follows trivially that $F[\ ] \overset{\hat{\nu}}{\Rightarrow} G'[\ ]$.

Before proceeding to extend the definition of $F[\ ] \xrightarrow[\mu]{\nu} F'[\ ]$ to multiple-length derivations we prove one more result which will be required later:

Proposition 4.6

If $F[\ ] \xrightarrow[\mu]{\nu} G[\ ]$ and $G[\ ] \overset{s}{\Rightarrow} G'[\ ]$, where $G'[b] \equiv C$, then $F[\ ] \overset{\hat{\nu}s}{\Rightarrow} C'[\ ]$.

Proof

By structural induction on $F[\ ]$; we may assume that $G'[b] \equiv K$ and have to consider the seven cases for $F[b]$ as listed in the proof of lemma 4.2.

(a) $F[b] \equiv C$.

The result is trivial in this case.

(b) $F[b] \equiv b$.

Here $F[\ ] \xrightarrow{\nu}_{\mu} G[\ ]$ implies that $x \xrightarrow{\nu} G[y]$ whenever $x \xrightarrow{\mu} y$; it follows that $\mu = \nu$ and $G[b] \equiv b$. Then, as $G[\ ] \overset{s}{\Rightarrow} G'[\ ]$ it follows that $x \overset{s}{\Rightarrow} G'[x]$ for all behaviours x, that is $x \overset{s}{\Rightarrow} K$ for all x. This is clearly impossible, so the assumptions of the proposition cannot hold in this case; hence there is nothing to prove.

(c) $F[b] \equiv \gamma.F'[b]$ for $\gamma \in \Lambda \cup \{\tau\}$.

Here we have $\gamma.F'[x] \xrightarrow{\nu} G[y]$ whenever $x \xrightarrow{\mu} y$, so $\nu = \gamma$ and $F'[x] \equiv G[y]$. This can occur only when F' is constant; hence F is constant and the result follows immediately.

In all the remaining cases the result is obtained using the structural-inductive hypothesis.

This completes the structural induction, hence the proposition holds for all contexts F.

The next step is to extend the definition of "causing" to multiple-length derivations; the obvious definition to use would be one analogous to that for $F[\ ] \xrightarrow{\nu}_{\mu} F'[\ ]$ with s and t (in Λ^*) in place of ν and μ, and $\Rightarrow$ instead of $\rightarrow$ but it is found that it is difficult to prove certain properties of this definition which are needed later due to the fact that in the derivation $A \overset{s}{\Rightarrow} B$ it is not known where τ-actions occur. Hence the definition chosen is an inductive one based on the definition of $F[\ ] \xrightarrow{\nu}_{\mu} F'[\ ]$.

Definition

For $s \in \Lambda^+$ and $t \in \hat{\Lambda}^*$, $F[\ ] \overset{t}{\underset{s}{\Rightarrow}} F'[\ ]$ if and only if there exists a sequence of contexts

$$F_0[\ ],\ F'_0[\ ] \ldots,\ F_n[\ ],\ F'_n[\ ]$$

such that F_0 is F and F'_n is F', with $s = \lambda_1 \ldots \lambda_n$ such that

(i) $F_i[\ ] \overset{t_i}{\Rightarrow} F'_i[\ ]$ for $i \in \{0, \ldots, n\}$

(ii) $F'_{i-1}[\ ] \underset{\rho_i}{\overset{\lambda_i}{\rightarrow}} F_i[\ ]$ for $i \in \{1, \ldots, n\}$

and $t = t_0 \hat{\rho}_1 t_1 \ldots \hat{\rho}_n t_n$.

It should be noted that the above definition does not include the case $s = \varepsilon$. If this case were included we would have $F[\ ] \overset{t}{\underset{\varepsilon}{\Rightarrow}} F'[\ ]$ defined to mean $F[\ ] \overset{t}{\Rightarrow} F'[\ ]$. However for the definition to be of any practical use regarding "causing" it is essential that if $F[\ ] \overset{t}{\underset{s}{\Rightarrow}} F'[\ ]$ then $A \overset{s}{\Rightarrow} B$ implies $F[A] \overset{t}{\Rightarrow} F'[B]$. Unfortunately this does not hold for $s = \varepsilon$ using such a definition of $F[\ ] \overset{t}{\underset{\varepsilon}{\Rightarrow}} F'[\ ]$. For example, let

$F[b] \equiv \alpha.\beta.b$ and
$F'[b] \equiv \beta.b$,

then $F[\ ] \overset{\alpha}{\Rightarrow} F'[\ ]$ so it would follow that $F[\ ] \overset{\alpha}{\underset{\varepsilon}{\Rightarrow}} F'[\ ]$. Then defining

$$A \equiv \alpha.\mathrm{NIL} + \tau.\mathrm{NIL}$$

we have $A \overset{\tau}{\Rightarrow} \mathrm{NIL}$, hence it should be provable that $F[A] \overset{\alpha}{\Rightarrow} F'[\mathrm{NIL}]$, that is that $\alpha.\beta.(\alpha.\mathrm{NIL} + \tau.\mathrm{NIL}) \overset{\alpha}{\Rightarrow} \beta.\mathrm{NIL}$ which is clearly untrue.

We shall eventually wish to prove a result similar to lemma 4.2 for multiple derivations, but the above definition of $\overset{t}{\underset{s}{\Rightarrow}}$ is not

adequate for this. Consider for example the context F defined by $F[b] \equiv b[\beta/\alpha]$ and let $A \equiv \tau.\alpha.NIL$. Then $F[A] \overset{\varepsilon}{\Rightarrow} (\alpha.NIL)[\beta/\alpha]$, so the lemma would imply that either $F[\ ] \overset{\varepsilon}{\Rightarrow} F'[\ ]$ for some F', which clearly is not true, or $F[\ ] \overset{\varepsilon}{\Rightarrow} F'[\ ]$, $A \overset{\zeta}{\Rightarrow} B$ and $F'[B] \equiv (\alpha.NIL)[\beta/\alpha]$. As this has not been defined for $s=\varepsilon$ and A's only derivations are $A \overset{\tau}{\Rightarrow} A$, $A \overset{\zeta}{\Rightarrow} \alpha.NIL$ and $A \overset{\alpha}{\Rightarrow} NIL$ we must have $s=\alpha$ and $B \equiv NIL$. However $F[\ ] \overset{\varepsilon}{\underset{\alpha}{\Rightarrow}} F'[\ ]$ is not true in this example, hence the lemma cannot hold unless we allow unobservable actions to "cause" derivations. As explained above the case $s=\varepsilon$ could not be included in the definition; the alternative we choose is to extend it to include $s=\tau$ (that is, considering only non-null unobservable actions).

Definition

$F[\ ] \overset{t}{\underset{\tau}{\Rightarrow}} F'[\ ]$ if and only if

$$F[\ ] \overset{t_0}{\Longrightarrow} F_0[\ ] \overset{\mu}{\underset{\tau}{\rightarrow}} F_1[\ ] \overset{t_1}{\Rightarrow} F'[\ ]$$

with $t=t_0\hat{\mu}t_1$.

Having defined the concept of $F[\ ] \overset{t}{\underset{s}{\Rightarrow}} F'[\ ]$ we must prove that it is a "causing" relation like $F[\ ] \overset{\nu}{\underset{\mu}{\rightarrow}} F'[\ ]$, that is that if $F[\ ] \overset{t}{\underset{s}{\Rightarrow}} F'[\ ]$ and $A \overset{s}{\Rightarrow} B$ then $F[A] \overset{t}{\Rightarrow} F'[B]$.

Proposition 4.7

If $A \overset{s}{\Rightarrow} B$ for $s \in \Lambda^+ \cup \{\tau\}$ and $F[\ ] \overset{t}{\underset{s}{\Rightarrow}} F'[\ ]$ then $F[A] \overset{t}{\Rightarrow} F'[B]$.

Proof

It is sufficient to prove the result for $|s|=1$, for from the definition of $F[\ ] \overset{t}{\underset{s}{\Rightarrow}} F'[\ ]$ it is easy to see that if $s=\lambda_1 \ldots \lambda_n$

then $F[\ ] \overset{t_0}{\underset{\lambda_1}{\Rightarrow}} F_1[\ ] \ldots F_{n-1}[\ ] \overset{t_n}{\underset{\lambda_n}{\Rightarrow}} F'[\ ]$ with $t=t_1 \ldots t_n$, and using n applications of the result for $|s|=1$ the required general result is obtained. The proof makes use of structural induction on contexts and in order to be able to treat the case $F[b] \equiv G[b]+C$ we use a slightly stronger inductive hypothesis than is necessary for the final result, proving also that the derivation $F[A] \overset{t}{\Rightarrow} F'[B]$ is non-null if $t=\varepsilon$.

Hence we suppose that $A \overset{\mu}{\rightarrow} B$ and prove by structural induction on F that if $F[\ ] \overset{t}{\underset{\mu}{\Rightarrow}} F'[\ ]$ then $F[A] \overset{t}{\Rightarrow} F'[B]$ and furthermore that this derivation is non-null even if $t=\varepsilon$. The definition of $F[\ ] \overset{t}{\underset{\mu}{\Rightarrow}} F'[\ ]$ gives us:

$$F[\ ] \overset{t_0}{\Rightarrow} F_0[\ ] \overset{\nu}{\underset{\mu}{\rightarrow}} F_1[\ ] \overset{t_1}{\Rightarrow} F'[\ ]$$

with $t=t_0 \hat{\nu} t_1$. For the structural induction on F there are seven cases to consider, as listed in the proof of lemma 4.2.

(a) $F[b] \equiv C$.
Here as $F[\ ] \overset{t_0}{\Rightarrow} F_0[\ ]$ we have $F_0[b] \equiv C_0$ for some C_0 such that $C \overset{t_0}{\Rightarrow} C_0$ and as $F_0[\ ] \overset{\nu}{\underset{\mu}{\rightarrow}} F_1[\ ]$ it follows that $F_1[b] \equiv C_1$ with $C_0 \overset{\nu}{\rightarrow} C_1$. Hence $F[A] \overset{t_0 \hat{\nu}}{\Rightarrow} F_1[B]$ non-nully and as $F_1[\ ] \overset{t_1}{\Rightarrow} F'[\ ]$ it follows that $F[A] \overset{t}{\Rightarrow} F'[B]$ in a non-null derivation, as required.

(b) $F[b] \equiv b$.
Here $F[\ ] \overset{t_0}{\Rightarrow} F_0[\ ]$ implies that $x \overset{t_0}{\Rightarrow} F_0[x]$ for all behaviours x, hence $t_0=\varepsilon$ and $F_0 \equiv F$. Hence $F[\ ] \overset{\nu}{\underset{\mu}{\rightarrow}} F_1[\ ]$, which means that whenever $x \overset{\mu}{\rightarrow} y$ then $F[x] \overset{\nu}{\rightarrow} F_1[y]$, that is $x \overset{\nu}{\rightarrow} F_1[y]$. This can only be valid if $F_1[b] \equiv b$ and $\mu=\nu$. Now as $F_1[\ ] \overset{t_1}{\Rightarrow} F'[\ ]$ the same argument as above shows that $t_1=\varepsilon$ and $F'[b] \equiv b$. Hence $F[\ ] \overset{t}{\underset{\mu}{\Rightarrow}} F'[\ ]$ can occur in this case only when $t=\hat{\mu}$ and $F'[b] \equiv b$. This means that all we need prove is that whenever $A \overset{\mu}{\rightarrow} B$ then $A \overset{\mu}{\rightarrow} B$, which is trivially true.

(c) $F[b] \equiv \aleph.G[b]$ for $\aleph \in A \cup \{\varepsilon\}$.

In this case $\aleph.G[\ ] \overset{t_0}{\Rightarrow} F_0[\ ]$ for which there are two cases to consider,

either $t_0 = \hat{\aleph} t_0'$ and $F[\ ] \xrightarrow{\aleph} G[\ ] \overset{t_0'}{\Rightarrow} F_0[\ ]$, in which case, defining $t' = t_0' \hat{\ } \nu \hat{\ } t_1$ we have $G[\ ] \overset{t'}{\underset{\mu}{\Rightarrow}} F'[\ ]$ and by the structural-inductive hypothesis we have $G[A] \overset{t'}{\Rightarrow} F'[B]$ and hence $F[A] \overset{t}{\Rightarrow} F'[B]$ in a non-null derivation as required;

or $t_0 = \varepsilon$ and $F_0[b] \equiv F[b]$, so $F[\ ] \overset{\nu}{\underset{\mu}{\Rightarrow}} F_1[\ ]$. This implies that whenever $x \xrightarrow[\mu]{} y$ then $\aleph.G[x] \xrightarrow{\nu} F_1[y]$, and hence it must be the case that $\nu = \aleph$ and $G[x] \equiv F_1[y]$. This can hold only if F_1 and G are the same constant context, and so $F[b] \equiv \aleph.C$ and $F_1[b] \equiv C$ for some behaviour C. Hence $F[A] \xrightarrow{\nu} F_1[B]$ and as $t_0 = \varepsilon$ it follows that $t = \hat{\nu} t_1$ giving $F[A] \overset{t}{\Rightarrow} F'[B]$ non-nully as required.

(d) $F[b] \equiv G[b] \backslash \alpha$.

Here it is not difficult to prove (by splitting $F[\ ] \overset{t_0}{\Rightarrow} F_0[\ ] \overset{\nu}{\underset{\mu}{\Rightarrow}} F_1[\ ] \overset{t_1}{\Rightarrow} F'[\ ]$ into single derivations) that α does not occur in the string t and that $G[\ ] \overset{t}{\underset{\mu}{\Rightarrow}} G'[\ ]$ where G' is such that $F'[b] \equiv G'[b] \backslash \alpha$. Then by the structural-inductive hypothesis we conclude that $G[A] \overset{t}{\Rightarrow} G'[B]$ in a non-null derivation and as α does not occur in t it follows that $F[A] \overset{t}{\Rightarrow} F'[B]$ in a non-null derivation as required.

(e) $F[b] \equiv G[b] + C$.

In this case it can be shown that either $G[\ ] \overset{t}{\underset{\mu}{\Rightarrow}} F'[\ ]$ or $C \overset{t}{\Rightarrow} C'$ non-nully and $F'[b] \equiv C'$. In the former case the structural-inductive hypothesis gives $G[A] \overset{t}{\Rightarrow} F'[B]$ (in a non-null derivation) and in the second case $C \overset{t}{\Rightarrow} F'[B]$ (in a non-null derivation). In either case it follows that $F[A] \overset{t}{\Rightarrow} G'[B]$ non-nully as required.

(f) $F[b] \equiv G[b]|C$.

Here it must be the case that $F_0[b] \equiv G_0[b]|C_0$ for some G_0 and C_0 such that $G[\ ] \overset{u_0}{\Rightarrow} G_0[\ ]$ and $C \overset{v_0}{\rightarrow} C_0$ for appropriate u_0 and v_0 in A^*. Then $F_0[\ ] \overset{\nu}{\underset{\mu}{\rightarrow}} F_1[\ ]$ can occur in three ways,

either $G_0[\ ] \overset{\nu}{\underset{\mu}{\rightarrow}} G_1[\ ]$ and $F_1[b] \equiv G_1[b]|C_0$,

or $\nu = \tau$ and $G_0[\ ] \overset{\lambda}{\underset{\mu}{\rightarrow}} G_1[\ ]$ and $C_0 \overset{\lambda}{\rightarrow} C_1$ for some $\lambda \in A$ and $F_1[b] \equiv G_1[b]|C_1$,

or $G_0[b] \equiv K$, $C_0 \overset{\nu}{\rightarrow} C_1$ and $F_1[b] \equiv K|C$.

In the first two cases the structural inductive hypothesis is applied to G in a manner similar to before; in the third case it can be seen that $F_0[A] \overset{\hat{\nu}}{\rightarrow} F_1[B]$ and as $F[\ ] \overset{t_0}{\Rightarrow} F_0[\ ]$ and $F_1[\ ] \overset{t_1}{\Rightarrow} F'[\ ]$ it follows that $F[A] \overset{t}{\rightarrow} F'[B]$ in a non-null derivation as required.

(g) $F[b] \equiv G[b][S]$.

This case is treated in a similar manner to (d).

This completes the structural induction and hence the proof of the proposition.

Having defined $F[\ ] \overset{t}{\underset{s}{\Rightarrow}} F'[\ ]$ for $s \in A^+ \cup \{\tau\}$ we are able to prove two propositions which express the essential results from propositions 4.3, 4.4, 4.5 and 4.6 in forms which are more convenient for later use.

Proposition 4.8

If $F[\ ] \overset{\nu}{\underset{\tau}{\Rightarrow}} G[\ ]$ and $G[\ ] \overset{t}{\underset{s}{\Rightarrow}} G'[\ ]$ then $F[\ ] \overset{\nu t}{\underset{s}{\Rightarrow}} G'[\ ]$.

Proof

From the definition of $G[\ ] \overset{t}{\underset{s}{\Rightarrow}} G'[\ ]$ we have either

$G[\ ] \overset{t_0}{\Rightarrow} G_0[\ ] \overset{\nu}{\underset{\pi}{\rightarrow}} G_1[\ ] \overset{t_1}{\Rightarrow} G'[\ ]$

in the case $s=\pi\epsilon\Lambda\cup\{\tau\}$, with $t=t_0\hat{\nu}t_1$, or

$G[\ ] \overset{t_0}{\Rightarrow} G_0[\ ] \overset{\nu}{\underset{\pi}{\rightarrow}} G_1[\ ] \overset{t_1}{\underset{s_1}{\Rightarrow}} G'[\ ]$

in the case $|s|>1$, with $s=\pi s_1$ and $t=t_0\hat{\nu}t_1$. In either case it will be enough to prove that $F[\ ] \overset{\hat{\nu}t_0\hat{\nu}}{\underset{\pi}{\Longrightarrow}} G_1[\ ]$ so we can treat the two cases together. We split the proof into two cases dependant upon whether $\nu=\tau$:

(i) If $\nu\neq\tau$ then proposition 4.3 implies that $F[\ ] \overset{\nu}{\rightarrow} G[\ ]$, so $F[\ ] \overset{\nu t_0}{\Longrightarrow} G_0[\ ] \overset{\nu}{\underset{\pi}{\rightarrow}} G_1[\ ]$ and $F[\ ] \overset{\hat{\nu}t_0\hat{\nu}}{\Longrightarrow} G_1[\ ]$ as required.

(ii) In the case $\nu=\tau$ we first look at $G[\ ] \overset{t_0}{\Rightarrow} G_0[\ ]$. This can occur in two ways,

<u>either</u> $t_0=\varepsilon$ and $G_0[b] \equiv G[b]$, in which case $F[\ ] \overset{\tau}{\underset{\tau}{\rightarrow}} G[\ ]$ and $G[\ ] \overset{\nu}{\underset{\pi}{\rightarrow}} G_1[\ ]$ implies by proposition 4.4 that $F[\ ] \overset{\hat{\nu}}{\underset{\pi}{\Rightarrow}} G_1[\ ]$ as required;

<u>or</u> $G[\ ] \overset{\sigma}{\rightarrow} G_0[\ ]$ for some $\sigma\epsilon(\Lambda\cup\{\tau\})^+$ and $t_0=\hat{\sigma}$. In this case we can write $\sigma=\theta\sigma'$, so $G[\ ] \overset{\theta}{\rightarrow} H[\ ] \overset{\sigma'}{\rightarrow} G_0[\ ]$. Applying proposition 4.5 to $F[\ ] \overset{\tau}{\underset{\tau}{\rightarrow}} G[\ ]$ and $G[\ ] \overset{\theta}{\rightarrow} H[\ ]$ we obtain $F[\ ] \overset{\hat{\theta}}{\Rightarrow} H[\ ]$ and as $H[\ ] \overset{\sigma'}{\rightarrow} G_0[\ ]$ and $t_0=\hat{\theta}\hat{\sigma'}$ this gives $F[\ ] \overset{t_0}{\Rightarrow} G_0[\ ] \overset{\nu}{\underset{\pi}{\rightarrow}} G_1[\ ]$, so $F[\ ] \overset{t_0\hat{\nu}}{\underset{\pi}{\Longrightarrow}} G_1[\ ]$ as required.

In all cases we have shown that $F[\ ] \overset{t_0\hat{\nu}}{\underset{\pi}{\Longrightarrow}} G_1[\ ]$, hence $F[\ ] \overset{t}{\underset{s}{\Rightarrow}} G'[\ ]$ as required.

<u>Proposition 4.9</u>

If $F[\ ] \overset{t}{\underset{s}{\Rightarrow}} F'[\ ]$ and F' is constant then $F[\ ] \overset{t}{\Rightarrow} F'[\ ]$.

Proof

By induction on $|s|$. As $s \in \Lambda^{+}\cup\{\tau\}$ the base step is $|s|=1$. In this case $F[\ ] \stackrel{t_0}{\Longrightarrow} F_0[\ ] \stackrel{\nu}{\rightarrow} F_1[\ ] \stackrel{t_1}{\Longrightarrow} F'[\ ]$ with $t=t_0\hat{}\nu\hat{}t_1$, so applying proposition 4.6 we obtain $F_0[\ ] \stackrel{\hat{\nu}t_1}{\Longrightarrow} F'[\ ]$, and hence $F[\ ] \stackrel{t}{\Rightarrow} F'[\ ]$ as required.

For the inductive step we consider $s=\lambda s'$ and assume that the result holds for s'. $F[\ ] \stackrel{t}{\underset{s'}{\Rightarrow}} F'[\ ]$ splits into $F[\ ] \stackrel{t_0}{\Longrightarrow} F_0[\ ] \stackrel{\nu}{\underset{\lambda}{\rightarrow}} F_1[\ ] \stackrel{t_1}{\underset{s'}{\Rightarrow}} F'[\ ]$ so applying the inductive hypothesis we deduce that $F_1[\ ] \stackrel{t_1}{\Rightarrow} F'[\ ]$ then by proposition 4.6 $F_0[\ ] \stackrel{\hat{\nu}t_1}{\Longrightarrow} F'[\ ]$, so $F[\ ] \stackrel{t}{\Rightarrow} F'[\ ]$ as required, completing the inductive step and hence the proof of the proposition.

Our next aim is to prove a result analogous to lemma 4.2 for the multiple-length "causing" relation; it is found that a direct proof is extremely lengthy and involved so we first prove a slightly different proposition using $\rightarrow$ instead of $\Rightarrow$ then deduce the desired lemma from this.

Proposition 4.10

If $F[A] \stackrel{\sigma}{\rightarrow} D$ for $\sigma \in (\Lambda\cup\{\tau\})^{*}$, and $u=\hat{\sigma}$ then either

(i) $F[\ ] \stackrel{u}{\underset{s}{\Rightarrow}} F'[\ ]$ for some $s \in \Lambda^{+}\cup\{\tau\}$ with $A \stackrel{s}{\Rightarrow} B$ and $D \equiv F'[B]$; or

(ii) $F[\ ] \stackrel{u}{\Rightarrow} F'[\ ]$ and $D \equiv F'[A]$.

Proof

By induction on $|\sigma|$. For the base step, $\sigma=\epsilon$, we have $u=\epsilon$ and $D \equiv F[A]$, and since it is trivially true that $F[\ ] \stackrel{\epsilon}{\Rightarrow} F[\ ]$ the second result holds with $F' \equiv F$.

For the inductive step at $|\sigma|=n+1$ we let $\sigma=\nu\sigma'$ with $|\sigma'|=n$, and let $u'=\widehat{\sigma'}$. From the assumption $F[A] \stackrel{\sigma}{\rightarrow} D$ it can be deduced that

$$F[A] \stackrel{\nu}{\rightarrow} X \stackrel{\sigma'}{\rightarrow} D$$

for some behaviour X. Then applying lemma 4.2 to the derivation $F[A] \stackrel{\nu}{\rightarrow} X$ there are two cases to consider:

(i) $F[\ ] \stackrel{\nu}{\underset{\mu}{\rightarrow}} G[\ ]$ with $A \stackrel{\mu}{\rightarrow} C$ and $X \equiv G[C]$, in which case $G[C] \stackrel{\sigma'}{\rightarrow} D$, so applying the inductive hypothesis

either $G[\ ] \stackrel{u'}{\underset{t}{\Rightarrow}} F'[\ ]$ for some $t \in A^{+} \cup \{\tau\}$, with $C \stackrel{t}{\Rightarrow} B$ and $D \equiv F'[B]$. If $\mu \neq \tau$ we have $F[\ ] \stackrel{\nu}{\underset{\mu}{\rightarrow}} G[\ ] \stackrel{u'}{\underset{t}{\Rightarrow}} F'[\ ]$ so (by the definition) $F[\ ] \stackrel{u}{\underset{s}{\Rightarrow}} F'[\ ]$ where $s=\mu t$, and as $A \stackrel{\mu}{\rightarrow} C \stackrel{t}{\Rightarrow} B$, it follows that $A \stackrel{s}{\Rightarrow} B$ as required. Otherwise, if $\mu=\tau$, proposition 4.8 implies that $F[\ ] \stackrel{u}{\underset{s}{\Rightarrow}} F'[\ ]$ where $s=t$ and, as $A \stackrel{s}{\rightarrow} C \stackrel{t}{\Rightarrow} B$, it follows that $A \stackrel{s}{\Rightarrow} B$ as required;

or $G[\ ] \stackrel{u'}{\Rightarrow} F'[\ ]$ and $D \equiv F'[C]$, so $F[\ ] \stackrel{\nu}{\underset{\mu}{\rightarrow}} G[\ ] \stackrel{u'}{\Rightarrow} F'[\ ]$, hence (by the definition) $F[\ ] \stackrel{u}{\underset{\mu}{\Rightarrow}} F'[\ ]$, and $A \stackrel{\mu}{\Rightarrow} C$ as required.

(ii) $F[\ ] \stackrel{\nu}{\rightarrow} G[\ ]$ with $X \equiv G[A]$, in which case $G[A] \stackrel{\sigma'}{\rightarrow} D$, and applying the inductive hypothesis it follows that

either $G[\ ] \stackrel{u'}{\underset{t}{\Rightarrow}} F'[\ ]$ and $A \stackrel{t}{\Rightarrow} B$ with $D \equiv F'[B]$, in which case $F[\ ] \stackrel{u}{\underset{s}{\Rightarrow}} F'[\ ]$ as required;

or $G[\ ] \stackrel{u'}{\Rightarrow} F'[\ ]$ and $D \equiv F'[A]$, when $F[\ ] \stackrel{u}{\Rightarrow} F'[\ ]$ satisfying the second clause of the proposition.

This completes the inductive step and hence the proof of the proposition.

The above proposition involved both $\rightarrow$ and $\Rightarrow$ derivations, and although it was expressed conveniently for proof it will be found that a neater result is needed later. Hence we present the following corollary.

Corollary 4.11

If $F[A] \stackrel{t}{\Rightarrow} D$ for some $t \in \Lambda^*$ then either

(i) $F[\ ] \stackrel{t}{\Rightarrow} F'[\ ]$ for some $s \in \Lambda^+_\nu(\tau)$ with $A \stackrel{s}{\Rightarrow} B$ for some behaviour B with $D \equiv F'[B]$; or

(ii) $F[\ ] \stackrel{t}{\Rightarrow} F'[\ ]$ and $D \equiv F'[A]$.

Proof

If $F[A] \stackrel{t}{\Rightarrow} D$ then $F[A] \stackrel{\sigma}{\rightarrow} D$ for some string $\sigma \in (\Lambda_\nu(\tau))^*$ such that $t = \hat{\sigma}$; hence the result follows immediately from proposition 4.10.

A direct proof of this corollary would be much longer than the proof of proposition 4.10 since an induction on $|t|$ would involve $F[A] \stackrel{\lambda}{\Rightarrow} X$ in the inductive step and as this would have to be expanded to $F[A] \xrightarrow{\tau^* \lambda \tau^*} X$ it would not be possible to apply proposition 4.2 without splitting the derivation further to remove the unobservable actions.

We are now ready to prove the central theorem of this chapter, for which we need a measure on derivation strings to which an induction technique can be applied; after proving the theorem we shall need to investigate more closely the assumptions made, in order to choose an appropriate measure and a class of contexts for which the conditions are satisfied. The second condition refers to "F and all its possible derivatives"; a possible derivative of F means any context G such that either $F[\ ] \stackrel{t}{\Rightarrow} G[\ ]$ or $F[\ ] \stackrel{t}{\underset{s}{\Rightarrow}} G[\ ]$ for some s and t. $\mathbf{N}$ denotes the set of non-negative integers $\{0,1,\ldots\}$.

Theorem 4.12

For any function $\#:\Lambda^*_v \cup \{\tau\} \to \mathbb{N}$, if a context F satisfies

(a) if $F[\ ] \overset{t}{\underset{\tau}{\Rightarrow}} F'[\ ]$ then $\#s < \#t$ or F' is constant; and

(b) F and all its possible derivatives preserve $\approx_k$ for all $k > 0$,

then if $A_1 \approx F[A_1]$ and $A_2 \approx F[A_2]$ then $A_1 \approx A_2$.

Proof

We show by induction on k that under the assumptions of the theorem $A_1 \approx_k A_2$ for all $k > 0$. The result holds trivially for k=0; for the inductive step we prove it for k+1 assuming that it holds for k.

We need to prove that whenever $A_1 \overset{t}{\Rightarrow} B_1$ then $A_2 \overset{t}{\Rightarrow} B_2$ for some B_2 such that $B_1 \approx_k B_2$. To do this we use induction on #t. For the base step, #t=0, as $A_1 \overset{t}{\Rightarrow} B_1$ and $A_1 \approx F[A_1]$ we can deduce $F[A_1] \overset{t}{\Rightarrow} D \approx_k B_1$. Applying corollary 4.11 either

(i) $F[\ ] \overset{t}{\underset{\tau}{\Rightarrow}} F'[\ ]$ with $A_1 \overset{s}{\rightarrow} C_1$ and $D \equiv F'[C_1]$, in which case by assumption (a) either $\#s < \#t$ or F' is constant. As #t=0 the latter must be true, so $F'[b] \equiv D$. Then applying proposition 4.9 we obtain $F[\ ] \overset{t}{\rightarrow} F'[\ ]$, hence $F[A_2] \overset{t}{\Rightarrow} F'[A_2] \equiv D$, and as $A_2 \approx F[A_2]$ it follows that $A_2 \overset{t}{\rightarrow} B_2$ for some agent B_2 such that $D \approx_k B_2$. As $D \approx_k B_1$ also, it can be concluded that $B_1 \approx_k B_2$ as required; or

(ii) $F[\ ] \overset{t}{\Rightarrow} F'[\ ]$ and $D \equiv F'[A_1]$, in which case $F[A_2] \overset{t}{\Rightarrow} F'[A_2]$. Then as $A_2 \approx F[A_2]$ it follows that $A_2 \overset{t}{\Rightarrow} B_2$ for some $B_2 \approx_k F'[A_2]$. By the inductive hypothesis $A_1 \approx_k A_2$ so by assumption (b) $F'[A_1] \approx_k F'[A_2]$, hence $B_2 \approx_k F'[A_1] \equiv D \approx_k B_1$ as required.

This completes the base step of the #t induction.

For the inductive step we assume the result holds for $\#t \leq n$ and prove it for $\#t=n+1$; the proof is identical to the base step proof above except that in case (i) it is possible that $\#s < \#t$, so $F'[\ ]$ need not be constant. Then, however, $\#s < n$ so, unless $s=\tau$, by the ($\#t$) inductive hypothesis, as $A_1 \stackrel{s}{\Rightarrow} C_1$, it follows that $A_2 \stackrel{s}{\Rightarrow} C_2 \approx_k C_1$. As $F[\ ] \stackrel{t}{\Rightarrow}_s F'[\ ]$, proposition 4.7 can be applied giving $F[A_2] \stackrel{t}{\Rightarrow} F'[C_2]$, and as F' preserves $\approx_k$ (by assumption (b)) we have $F'[C_2] \approx_k F'[C_1] \equiv D \approx_k B_1$. Then, as $A_2 \approx F[A_2]$ it follows that $A_2 \stackrel{t}{\Rightarrow} B_2 \approx_k F'[C_2] \approx_k B_1$ as required. This argument cannot be used if $s=\tau$ since the ($\#t$) inductive hypothesis cannot be applied to $A_1 \stackrel{\tau}{\Rightarrow} C_1$. However, in this case applying proposition 4.7 to $\tau.A_2 \stackrel{\tau}{\Rightarrow} A_2$ gives $F[\tau.A_2] \stackrel{t}{\Rightarrow} F'[A_2]$. Then as $A_2 \approx_{k+1} \tau.A_2$ and F preserves $\approx_{k+1}$ we have $F[A_2] \approx_{k+1} F[\tau.A_2]$, and hence $A_2 \approx_{k+1} F[\tau.A_2]$, so $A_2 \stackrel{t}{\Rightarrow} B_2$ for some behaviour B_2 such that $B_2 \approx_k F'[A_2]$. The inductive hypothesis gives $A_1 \approx_k A_2$, hence by assumption (b) $F'[A_1] \approx_k F'[A_2]$, so $B_2 \approx_k F'[A_1] \equiv D \approx_k B_1$ as required.

This completes the inductive proof that whenever $A_1 \stackrel{t}{\Rightarrow} B_1$ then $A_2 \stackrel{t}{\Rightarrow} B_2$ with $B_1 \approx_k B_2$; the same result can be proved with the roles of A_1 and A_2 interchanged, proving that $A_1 \approx_{k+1} A_2$ and completing the main inductive step of the proof of the theorem.

Having proved this theorem an investigation into the conditions (a) and (b) and contexts that satisfy them is necessary in order to be able to apply it to practical examples. Before proceeding to this we shall prove another property of the causing relation, namely that if $F[b] \equiv G[H[b]]$ and $F[\ ] \stackrel{t}{\Rightarrow}_s F'[\ ]$ then this "causing" can be split into two separate relations for the contexts G and H. As usual it is found easier to prove the result first for the single-step relation $\stackrel{\nu}{\rightarrow}_\mu$ and then extend it to the more general relation $\stackrel{t}{\Rightarrow}_s$ using properties of the definition.

Proposition 4.13

If $F[b] \equiv G[H[b]]$ and $F[\ ] \xrightarrow{\nu}_{\mu} F'[\ ]$ then either F' is constant or $G[\ ] \xrightarrow{\nu}_{\pi} G'[\ ]$ and $H[\ ] \xrightarrow{\pi}_{\mu} H'[\ ]$ for some $\pi \in \Lambda \vee \{\tau\}$ such that $F'[b] \equiv G'[H'[b]]$.

Proof

By structural induction on the context $G[\ ]$. There are seven cases to consider as in the proof of lemma 4.2:

(a) $G[b] \equiv C$.
Here F is constant, hence F' is constant.

(b) $G[b] \equiv b$.
In this case $F[b] \equiv H[b]$, so defining $H'[b] \equiv F'[b]$ we have $H[\ ] \xrightarrow{\nu}_{\mu} H'[\ ]$, and defining $G'[b] \equiv b$ it is trivially true that $G[\ ] \xrightarrow{\nu}_{\mu} G'[\ ]$ and $F'[b] \equiv G'[H'[b]]$.

(c) $G[b] \equiv \chi.G_0[b]$ $(\chi \in \Lambda \vee \{\tau\})$.
Here $F[b] \equiv \chi.G_0[H[b]]$, and as $F[\ ] \xrightarrow{\nu}_{\mu} F'[\ ]$ it follows that whenever $A \xrightarrow{\rho}_{\mu} B$ then $\chi.G_0[H[A]] \xrightarrow{\rho}_{\mu} F'[B]$, hence $\rho = \chi$ and $F'[B] \equiv G_0[H[A]]$. As this holds for all A and B satisfying $A \xrightarrow{\rho}_{\mu} B$ it follows that F' is constant.

(d) $G[b] \equiv G_0[b]\backslash\kappa$.
Here defining $F_0[b] \equiv G_0[H[b]]$ we have $F[b] \equiv F_0[b]\backslash\kappa$, and as $F[\ ] \xrightarrow{\nu}_{\mu} F'[\ ]$ it follows that $F_0[\ ] \xrightarrow{\nu}_{\mu} F'_0[\ ]$ for some F'_0 and $F'[b] \equiv F'_0[b]\backslash\kappa$, and that $\nu \neq \kappa$. Then applying the stuctural-inductive hypothesis either F'_0 is constant which implies that F' is constant, or $H[\ ] \xrightarrow{\pi}_{\mu} H'[\ ]$ and $G_0[\ ] \xrightarrow{\nu}_{\pi} G'_0[\ ]$ with $F'_0[b] \equiv G'_0[H'[b]]$. In this case defining $G'[b] \equiv G'_0[b]\backslash\kappa$ it can be seen that $G[\ ] \xrightarrow{\nu}_{\pi} G'[\ ]$ and $F'[b] \equiv G'[H'[b]]$ as required.

(e) $G[b] \equiv G_0[b]+C$.

In this case we have $F[b] \equiv G_0[H[b]]+C$, and as $F[\ ] \xrightarrow{\nu}_{\mu} F'[\ ]$ it follows that either $C \equiv F'[B]$ for all B in which case F' is constant, or $G_0[H[\ ]] \xrightarrow{\nu}_{\mu} F'[\ ]$, in which case the required result is obtained using the inductive hypothesis.

(f) $G[b] \equiv G_0[b]|C$.

Here defining $F_0[b] \equiv G_0[H[b]]$ we have $F[b] \equiv F_0[b]|C$, so $F[\ ] \xrightarrow{\nu}_{\mu} F'[\ ]$ gives three possibilities:

either $C \xrightarrow{\nu} D$ with $F_0[A] \equiv F_0[B]$ whenever $A \xrightarrow{\mu} B$, in which case it follows that F_0 is constant, and hence so is F';

or $\rho=\tau$ and, for some $\lambda \in \Lambda$, $F_0[\ ] \xrightarrow{\lambda}_{\mu} F'_0[\ ]$ and $C \xrightarrow{\bar{\lambda}} D$ and $F'[b] \equiv F'_0[b]|D$, when applying the inductive hypothesis either F'_0 is constant, which implies that F' is, or we obtain $H[\ ] \xrightarrow{\pi}_{\mu} H'[\ ]$ and $G_0[\ ] \xrightarrow{\lambda}_{\pi} G'_0[\ ]$ with $F'_0[b] \equiv G'_0[H'[b]]$. Then defining $G'[b] \equiv G'_0[b]|D$ we have $G[\ ] \xrightarrow{\tau}_{\pi} G'[\ ]$ and also $F'[b] \equiv G'[H'[b]]$ as required;

or $F_0[\ ] \xrightarrow{\nu}_{\mu} F'_0[\ ]$ and $F'[b] \equiv F'_0[b]|C$ in which case the required result is again deduced using the inductive hypothesis.

(g) $G[b] \equiv G_0[b][S]$.

The required result follows from the inductive hypothesis in this case with no difficulty.

In order to be able to use the above result in our investigation of which contexts satisfy the conditions (a) and (b) imposed in theorem 4.12 it will be convenient to extend it as usual to the non-single-length derivation case. This will be done by splitting $F[\ ] \overset{\epsilon}{\underset{s}{\Rightarrow}} F'[\ ]$ into single-length steps. Having done this it will be necessary to consider derivations of the form $F[\ ] \xrightarrow{\nu} F'[\ ]$ as well as $F[\ ] \xrightarrow{\nu}_{\mu} F'[\ ]$. Hence we first need a

result similar to proposition 4.13 for such derivations.

Proposition 4.14

If $F[b] \equiv G[H[b]]$ and $F[\ ] \xrightarrow{\nu} F'[\ ]$ then either

(i) $H[\ ] \xrightarrow{\mu} H'[\ ]$ and $G[\ ] \xrightarrow{\nu}_{\mu} G'[\ ]$ for some $\mu \in \Lambda \cup \{\tau\}$ with $F'[b] \equiv G'[H'[b]]$; or

(ii) $G[\ ] \xrightarrow{\nu} G'[\ ]$ and $F'[b] \equiv G'[H[b]]$.

Proof

By structural induction on G in a similar manner to the proof of proposition 4.13.

We are now ready to prove a result analogous to proposition 4.13 for derivations $F[\ ] \overset{t}{\underset{s}{\Rightarrow}} F'[\ ]$.

Proposition 4.15

If $F[b] \equiv G[H[b]]$ and $F[\ ] \overset{t}{\underset{\nu}{\Rightarrow}} F'[\ ]$ then either F' is constant or $G[\ ] \overset{t}{\underset{\sigma}{\Rightarrow}} G'[\ ]$ and $H[\ ] \overset{u}{\underset{s}{\Rightarrow}} H'[\ ]$ for some $\sigma \in \hat{\Lambda}^{+} \cup \{\tau\}$ such that $u = \hat{\sigma}$ with $F'[b] \equiv G'[H'[b]]$.

Proof

By induction on $|s|$. We will consider the base step in detail; the inductive step is treated similarly. For $|s|=1$ the definition of $F[\ ] \overset{t}{\underset{\nu}{\Rightarrow}} F'[\ ]$ gives

$$F[\ ] \overset{t_0}{\Rightarrow} F_0[\ ] \xrightarrow{\nu}_{\mu} F_1[\ ] \overset{t_1}{\Rightarrow} F'[\ ]$$

with $t=t_0\hat{}t_1$. The first step is to split the derivation $F[\ ] \overset{t_0}{\Rightarrow} F_0[\ ]$ into single-length steps to which proposition 4.14 can be applied. This gives two possibilities:

(i) $H[\ ] \overset{u_0}{\Rightarrow} H_0[\ ]$ and $G[\ ] \overset{t_0}{\underset{u_0}{\Rightarrow}} G_0[\ ]$ for some $u_0 \in A^+ \cup \{\tau\}$ with $F_0[b] \equiv G_0[H_0[b]]$, or

(ii) $G[\ ] \overset{t_0}{\Rightarrow} G_0[\ ]$ with $F_0[b] \equiv G_0[H[b]]$.

In the second case we can define $H_0[b] \equiv H[b]$ so that in either case we have $F_0[b] \equiv G_0[H_0[b]]$. Applying proposition 4.13 to $F_0[\ ] \overset{\nu}{\underset{\rho}{\rightarrow}} F_1[\ ]$ either F_1 is constant, in which case F' is constant and there is nothing more to prove, or $G_0[\ ] \overset{\nu}{\underset{\pi}{\rightarrow}} G_1[\ ]$ and $H_0[\ ] \overset{\pi}{\underset{\rho}{\rightarrow}} H_1[\ ]$ for some $\pi \in A \cup \{\tau\}$ with $F_1[b] \equiv G_1[H_1[b]]$. The next step is to split the derivation $F_1[\ ] \overset{t_1}{\Rightarrow} F'[\ ]$ into single steps and apply proposition 4.14 to these, again giving two possibilities:

(a) $H_1[\ ] \overset{u_1}{\Rightarrow} H'[\ ]$ and $G_1[\ ] \overset{t_1}{\underset{u_1}{\Rightarrow}} G'[\ ]$ for some $u_1 \in A^+ \cup \{\tau\}$ with $F'[b] \equiv G'[H'[b]]$, or

(b) $G_1[\ ] \overset{t_1}{\Rightarrow} G'[\ ]$ with $F'[b] \equiv G'[H_1[b]]$.

In the second case we can define $H'[b] \equiv H_1[b]$ to obtain $F'[b] \equiv G'[H'[b]]$ as required. It then remains to show that $G[\ ] \overset{t}{\underset{\sigma}{\Rightarrow}} G'[\ ]$ and $H[\ ] \overset{u}{\underset{\rho}{\Rightarrow}} H'[\ ]$ for appropriate σ and u. There are four cases to consider. We consider (ia) and (iib) in detail; the other two are treated similarly.

(ia) Here we have $G[\ ] \overset{t_0}{\underset{u_0}{\Rightarrow}} G_0[\ ] \overset{\nu}{\underset{\pi}{\rightarrow}} G_1[\ ] \overset{t_1}{\underset{u_1}{\Rightarrow}} G'[\ ]$ and $H[\ ] \overset{u_0}{\Rightarrow} H_0[\ ] \overset{\pi}{\underset{\rho}{\rightarrow}} H_1[\ ] \overset{u_1}{\Rightarrow} H'[\ ]$, so defining $\sigma = \hat{u}_0 \pi \hat{u}_1$ and $u = \hat{\sigma}$ the required results are obtained.

(iib) Here $G[\ ] \overset{t_0}{\Rightarrow} G_0[\ ] \overset{\nu}{\underset{\pi}{\rightarrow}} G_1[\ ] \overset{t_1}{\Rightarrow} G'[\ ]$ and $H[\ ] \overset{\pi}{\underset{\rho}{\rightarrow}} H'[\ ]$, so we need to define $\sigma = \pi$ and $u = \hat{\sigma}$.

This completes the proof for the base step of the induction. For the inductive step at $|s|=n+1$ $F[\ ] \stackrel{\epsilon}{\Rightarrow} F'[\ ]$ is expressed in the form $F[\ ] \stackrel{t_0}{\Rightarrow} F_0[\ ] \stackrel{\nu}{\rightarrow} F_1[\ ] \stackrel{t_1}{\Rightarrow} F'[\ ]$ with $|s'|=n$. The same technique is used as above, applying proposition 4.14 to $F[\ ] \stackrel{t_0}{\Rightarrow} F_0[\ ]$, proposition 4.13 to $F_0[\ ] \stackrel{\nu}{\rightarrow} F_1[\ ]$ and the inductive hypothesis to $F_1[\ ] \stackrel{t_1}{\Rightarrow} F'[\ ]$.

Having obtained this result we are ready to investigate the implications of the conditions (a) and (b) imposed in theorem 4.12. The first thing to consider is the measure $\#$. For the requirements of the theorem it was necessary for this to be defined on strings in $\Lambda^*\cup\{\tau\}$. However it will be convenient to define it on Λ^* imposing the constraint that $\#\sigma = \#\hat{\sigma}$ to extend it to strings in $(\Lambda\cup\{\tau\})^*$.

An obvious immediate choice for a measure to use would be the length $|s|$ of the string. However, considering the example introduced earlier and proved to be contracting,

$$F[b] \equiv \alpha.(b|\tilde{\beta}.NIL)\backslash L,$$

where $L=\{\alpha,\beta\}$ and defining

$$F_0[b] \equiv (b|\tilde{\beta}.NIL)\backslash L \text{ and}$$
$$F'[b] \equiv (b|NIL)\backslash L,$$

it can be seen that $F[\ ] \stackrel{\alpha}{\Rightarrow} F_0[\ ] \stackrel{\tau}{\rightarrow} F'[\ ]$, and hence that $F[\ ] \stackrel{\alpha}{\Rightarrow} F'[\ ]$, so with $\#s$ defined to be $|s|$ this example would not satisfy the conditions of theorem 4.12. In order for the theorem to serve any practical purpose it is essential that it can be used to show that examples of this complexity (at least!) are contracting; hence we need to use an alternative measure. In this example it can be seen that using the number of occurences of α in the string would give $\#s<\#t$ as desired in this case; indeed it can be verified that with this measure the context F as defined above

does satisfy the conditions for theorem 4.12. Furthermore, recalling that non-contracting transformations often involve "swallowing", and observing that here the α is not "swallowed" but in the similar example

$$F[b] \equiv \alpha.(b|\bar{\alpha}.NIL)\backslash\alpha,$$

where α is "swallowed", this measure does not satisfy the conditions for theorem 4.12 (for example $F[\ ] \overset{\alpha}{\Rightarrow} F'[\ ]$, where $F'[b] \equiv (b|NIL)\backslash\alpha$, it seems reasonable to use as a measure $\sharp$ the number of occurences of an arbitrary λ_0 in Λ, chosen so as not to be "swallowed".

Definition

The measure $\sharp:\Lambda^* \to \mathbb{N}$ is defined by

$$\begin{aligned} \sharp\varepsilon &= 0 \\ \sharp(\lambda_0 s) &= \sharp s+1 \\ \sharp(\lambda s) &= \sharp s, \text{ for } \lambda \neq \lambda_0. \end{aligned}$$

$\sharp v$ can be extended to $(A\cup\{\tau\})^*$ by defining $\sharp\sigma=\sharp\hat{\sigma}$.

For the proofs that follow we can regard λ_0 as a fixed element of Λ, and indeed the proofs could be repeated for every λ_0 in Λ, enabling us subsequently to treat λ_0 as arbitrary.

Having chosen a measure $\sharp$ the next aim is to find a class of contexts which satisfy condition (a) of theorem 4.12. We start by recalling that the context F defined by

$$F[b] \equiv \alpha.(b|\bar{\alpha}.NIL)\backslash\alpha$$

is not contracting and observe that for this context $F[\ ] \overset{\alpha}{\Rightarrow} F'[\ ]$

where

$$F'[b] \equiv (b|NIL)\backslash\alpha.$$

Furthermore, defining

$$G[b] \equiv \alpha.\beta.(b|\bar{\alpha}.NIL)\backslash\alpha,$$

it can easily be seen that $G[\ ] \overset{\alpha\beta}{\Rightarrow}_{\alpha} F'[\ ]$. This example and similar ones involving $(b|\bar{\alpha}.NIL)\backslash\alpha$ within other contexts suggest that we should eliminate contexts of the form $H[B|D[\ ]]$ from consideration when $\bar{\lambda_0} \in L_B$ (the sort of B). We shall also not allow relabelling of λ_0 anywhere within a context (if this were allowed we should have to insist that $\bar{\lambda} \notin L_B$ for any λ such that λ_0 may be relabelled to become λ, leading to unnecessary complications. It is not unreasonable to eliminate such relabelling since if λ_0 is relabelled in a context it will usually be possible to prove that the context is equivalent to another in which the relabelling does not occur. We are ready to prove a result about contexts satisfying the limitations just discussed.

Lemma 4.16

If a context G is not of the form $G[b] \equiv H[B|D[b]]$ with $\bar{\lambda_0} \in L_B$, and λ_0 is not relabelled anywhere within G then $G[\ ] \overset{s}{\Rightarrow}_{t} G'[\ ]$ implies that either $\#s \leq \#t$ or G' is constant.

Proof

As $G[\ ] \overset{s}{\Rightarrow}_{t} G'[\ ]$ can be split up into $G[\ ] \overset{s_1}{\Rightarrow}_{t_1} G_1[\ ] \ldots G_{n-1}[\ ] \overset{s_n}{\Rightarrow}_{t_n} G'[\ ]$ with $s=\lambda_1 \ldots \lambda_n$ and $t=t_1 \ldots t_n$ and $\#u + \#v = \#(uv)$ it will be adequate to prove the result for $|s|=1$. Furthermore in this case, unless $s=\lambda_0$ we have $\#s=0$ and it follows trivially that $\#s \leq \#t$. So it is necessary to consider only $s=\lambda_0$ in which case $\#s=1$ and the

proof involves showing that either $\#t \geq 1$ or G' is constant when $G[\ ] \overset{t}{\Rightarrow} G'[\ ]$. The definition of $G[\ ] \overset{t}{\Rightarrow} G'[\ ]$ gives $G[\ ] \overset{t_0}{\Rightarrow} G_0[\ ] \overset{\nu}{\rightarrow} G_1[\ ] \overset{t_1}{\Rightarrow} G'[\ ]$ with $t=t_0 \hat{} \nu t_1$. From the definition of $G[\ ] \overset{t_0}{\Rightarrow} G_0[\ ]$ it can be deduced that G_0 satisfies the constraints imposed on G, then by structural induction on G_0 it can be easily shown that $G_0[\ ] \overset{\nu}{\rightarrow} G_1[\ ]$ implies that either G_1 is constant or $\nu=\lambda_0$; in the first case it follows that G' is constant, and in the second case $\#\nu=1$, hence $\#t \geq 1$ as required.

It should be noted that the above lemma proved only that $\#s \leq \#t$ and not that $\#s<\#t$; this is only to be expected because, for example, in the context F studied earlier,

$$F[b] \equiv \alpha.(b|\vec{\alpha}.NIL)\backslash\alpha,$$

unless the arbitrarily-chosen λ_0 is equal to α, F satisfies the conditions imposed within the statement of the lemma. It is clear that the choice of λ_0 for any particular example such as the one above depends on the contexts H and D as in the statement of the lemma. As another example returning to the context G defined earlier by

$$G[b] \equiv \alpha.\beta.(b|\vec{\alpha}.NIL)\backslash\alpha,$$

it can be shown that this context is in fact contracting, although as already mentioned $G[\ ] \overset{\alpha\beta}{\underset{\alpha}{\Rightarrow}} F'[\ ]$ where

$$F'[b] \equiv (b|NIL)\backslash\alpha;$$

hence in this case α is not the appropriate λ_0 to choose. The reason for this lies in the "swallowing" of guards; a context appears to be contracting only if it is well-guarded by a guard which cannot be "swallowed". Hence the choice of λ_0 should be a guard which occurs in the context as well as being appropriate to

the non-"swallowing" requirements as imposed in lemma 4.16. Then in the above examples there is no appropriate λ_0 for F, but β is appropriate for G. We now need to show that the actual occurrence of λ_0 in the context, along with the result obtained from the "swallowing" restrictions in lemma 4.16 do in fact imply the required result that $\#s<\#t$ when $F[\ ] \overset{t}{\underset{s}{\Rightarrow}} F'[\ ]$ (and F' is not constant). To allow the occurence of λ_0 to occur effectively within either of the contexts H or D as in the statement of lemma 4.16 we specify that the context F must be of the form $F[b] \equiv G[\lambda_0.C[b]]$, with both G and C satisfying the conditions of the lemma. (Taking G to be the identity effectively puts the λ_0 guard within H, whereas taking C to be the identity puts it within D; however there is no need to limit the class of contexts under consideration by insisting that either G or C be the identity context).

As the context F[b] is of the form G[U[C[b]]] and we are considering $F[\ ] \overset{t}{\underset{s}{\Rightarrow}} F'[\ ]$ it is necessary to understand how this can be caused by relations of the form $G[\ ] \overset{t}{\underset{v}{\Rightarrow}} G'[\ ]$, $U[\ ] \overset{\hat{v}}{\underset{u}{\Rightarrow}} U'[\ ]$ and $C[\ ] \overset{\hat{u}}{\underset{s}{\Rightarrow}} C'[\ ]$; hence the need for proposition 4.15.

Theorem 4.17

Let $F[b] \equiv G[\lambda_0.C[b]]$ with neither G[b] nor C[b] of the form H[B|D[b]] with $\overline{\lambda_0} \in L_B$ and furthermore assume that λ_0 is not relabelled anywhere within the context F. Then whenever $F[\ ] \overset{t}{\underset{s}{\Rightarrow}} F'[\ ]$ it follows that either $\#s<\#t$ or F' is constant.

Proof

Define $V[b] \equiv \lambda_0 . C[b]$, then $F[b] \equiv G[V[b]]$. Then applying proposition 4.15 to $F[\ ] \overset{t}{\underset{s}{\Rightarrow}} F'[\ ]$ either F' is constant, in which case there is nothing more to prove, or $G[\ ] \overset{t}{\underset{\sigma}{\Rightarrow}} G'[\ ]$ and $V[\ ] \overset{v}{\underset{s}{\Rightarrow}} V'[\ ]$ for some $\sigma \in \Lambda^{+}\cup\{\tau\}$ such that $v=\hat{\sigma}$ and $F'[b] \equiv G'[V'[b]]$.

Define $U[b] \equiv \lambda_0.b$, then $V[b] \equiv U[C[b]]$. Applying proposition 4.15 to $V[\ ] \overset{v}{\Rightarrow} V'[\ ]$ either V' is constant, in which case F' is also constant and the proof is completed, or $U[\ ] \overset{v}{\Rightarrow} U'[\ ]$ and $C[\ ] \overset{u}{\Rightarrow} C'[\ ]$ for some $\theta \in \Lambda^+ \cup \{\tau\}$ such that $u=\hat{\theta}$ and $V'[b] \equiv U'[C'[b]]$.

Hence we have $G[\ ] \overset{t}{\Rightarrow} G'[\ ]$, $C[\ ] \overset{u}{\Rightarrow} C'[\ ]$ and $U[\ ] \overset{v}{\Rightarrow} U'[\ ]$, so applying lemma 4.16 to the first two of these we obtain $\#v < \#t$ or G' is constant, and $\#s < \#u$ or C' is constant. If either G' or C' is constant then it follows that F' is and there is nothing more to prove; hence we may assume that $\#s < \#u$ and $\#v < \#t$. We next consider $U[\ ] \overset{v}{\Rightarrow} U'[\ ]$. Letting A be any behaviour such that $A \overset{\theta}{\Rightarrow} B$ it follows that $U[A] \overset{v}{\Rightarrow} U'[B]$. Then as $U[A]=\lambda_0.A$ it follows that either $v=\lambda_0$ and $U'=U$ or $v=\lambda_0 v'$ for some v'. In the former case we have $U[A] \equiv U[B]$ whenever $A \overset{\theta}{\Rightarrow} B$, which is clearly not possible with U as defined, hence the latter case must hold. So whenever $A \overset{\theta}{\Rightarrow} B$ we have $\lambda_0.A \overset{\lambda_0 v'}{\Rightarrow} U'[B]$ and hence $A \overset{v'}{\Rightarrow} U'[B]$. This can occur only if $U'[b] \equiv b$ and $\theta=v'$, hence as $u=\hat{\theta}$ it follows that $\#u = \#v'$, and from $\#s < \#u$ it can be deduced that $\#s < \#v'$. Finally $\#t > \#v = \#\lambda_0 + \#v' = \#v'+1 > \#v' > \#s$, that is $\#s < \#t$ as required, completing the proof.

Having obtained a class of contexts that satisfy the first condition imposed in theorem 4.12 we now need to investigate the second condition, that is prove that certain contexts and all their possible derivatives preserve $\approx_k$ for all $k > 0$. As with the investigations of $\approx_c$ in [Mil4] it can be seen that unguarded ambiguity causes problems, for example define the context F by

$$F[b] \equiv \mathcal{A}.\mathrm{NIL}+b.$$

Then let $A \equiv \alpha.\mathrm{NIL}$ and $B \equiv \tau.\alpha.\mathrm{NIL}$, so $A \approx_2 B$. However $F[B] \overset{\varepsilon}{\Rightarrow} \alpha.\mathrm{NIL}$, so if F preserved $\approx_2$ we would have $F[A] \approx_2 F[B]$, and hence $F[A] \overset{\varepsilon}{\Rightarrow} C$ with $C \approx_1 \alpha.\mathrm{NIL}$. As $F[A]$ is stable its only ε-derivative

is F[A] so this would imply that $F[A] \approx_1 \alpha.NIL$ which is clearly untrue as F[A] has a β-derivative whereas $\alpha.NIL$ does not.

With Milner's $\approx_c$ investigation these problems occured only if the unguarded ambiguity was at the top level, so that the context G defined by

$$G[b] \equiv \alpha.F[b]$$

does preserve $\approx$ if F does (and indeed $\approx_k$ for any $k > 0$). However our condition (b) for theorem 4.12 insisted that all possible derivatives of a context should also preserve $\approx_k$, so G does not necessarily satisfy the requirements of the theorem (as we do not know all of its derivatives). Hence we shall require that if a context F can be expressed in the form $F[b] \equiv G[H[b]+C]$ then b is guarded within in H[b]. As this property will be used frequently we shall refer to it as "firm-guardedness".

Definition

A context F is said to be firmly guarded if whenever F can be expressed in the form $F[b] \equiv G[H[b]+C]$ then H[b] is of the form $U[\rho.V[b]]$.

In order to demonstrate that firmly-guarded contexts preserve $\approx_k$ for all k we begin by proving a stronger version of proposition 4.3 that holds for these contexts.

Lemma 4.18

If F is a firmly-guarded context and $F[\ ] \xrightarrow{\nu} F'[\ ]$ then $F[\ ] \overset{\hat{\nu}}{\Rightarrow} F'[\ ]$.

Proof

By proposition 4.3 $F[\ ] \xrightarrow{\nu}_{\tau} F'[\ ]$ implies that either $\nu = \tau$ or $F[\ ] \xrightarrow{\nu} F'[\ ]$; hence we already know that the result holds in all cases except $\nu = \tau$. All that has to be proved here is that if F is firmly-guarded and $F[\ ] \xrightarrow{\tau}_{\tau} F'[\ ]$ then $F[\ ] \overset{\varepsilon}{\Rightarrow} F'[\ ]$. We do this by structural induction on F. As usual there we consider the seven cases as listed in the proof of proposition 4.2:

(a) $F[b] \equiv C$.
In this case we have $C \xrightarrow{\tau} F'[B]$ for all B such that $A \xrightarrow{\tau} B$; since for any B there is an A such that $A \xrightarrow{\tau} B$ it follows that $C \xrightarrow{\tau} F'[B]$ for all behaviours B, that is $F[B] \xrightarrow{\tau} F'[B]$ for all B, so $F[\ ] \overset{\varepsilon}{\Rightarrow} F'[\ ]$ as required.

(b) $F[b] \equiv b$.
Here whenever $A \xrightarrow{\tau} B$ it follows that $A \xrightarrow{\tau} F'[B]$; hence it must be the case that $F'[b] \equiv b$, so it is trivially true that $F[\ ] \overset{\varepsilon}{\Rightarrow} F'[\ ]$.

(c) $F[b] \equiv \alpha.G[b]$.
In this case we have $\alpha.G[A] \xrightarrow{\tau} F'[B]$ whenever $A \xrightarrow{\varepsilon} B$. It follows that $\alpha = \tau$ and that $G[A] \equiv F'[B]$, which can occur only if G and F' are constant and $F' \equiv G$; hence $F[b] \equiv \tau.F'[b]$, so $F[\ ] \overset{\varepsilon}{\Rightarrow} F'[\ ]$.

(d) $F[b] \equiv G[b]\backslash\alpha$.
Here $F[\ ] \xrightarrow{\tau}_{\tau} F'[\ ]$ implies that $G[\ ] \xrightarrow{\tau}_{\tau} G'[\ ]$ and $F'[b] \equiv G'[b]\backslash\alpha$. Then by the inductive hypothesis, as G must be firmly-guarded it follows that $G[\ ] \overset{\varepsilon}{\Rightarrow} G'[\ ]$, and hence $F[\ ] \overset{\varepsilon}{\Rightarrow} F'[\ ]$ as required.

(e) $F[b] \equiv G[b]+C$.
This is the case that requires that F is

firmly-guarded. As $G[A]+C \xrightarrow{\tau} F'[B]$ whenever $A \xrightarrow{\tau} B$ it follows that either $C \xrightarrow{\tau} F'[B]$ for all B (with F' constant), when the result follows trivially, or $G[A] \xrightarrow{\tau} F'[B]$ whenever $A \xrightarrow{\tau} B$, that is $G[\ ] \xrightarrow[\tau]{} F'[\ ]$. As F is firmly guarded G is of the form $G[b] \equiv U[\mu.V[b]]$, and defining $W[b] \equiv \mu.V[b]$ we have $G[b] \equiv U[W[b]]$. Then applying proposition 4.15 gives two possibilities:

either F' is constant, equal to D, say, in which case $G[A] \xrightarrow{\tau} D$ whenever $A \xrightarrow{\tau} B$, so we have $G[\ ] \xrightarrow{\tau} D$, that is $G[\ ] \xrightarrow{\tau} F'[\ ]$, and $F[\ ] \overset{\xi}{\Rightarrow} F'[\ ]$ as required;

or $U[\ ] \xrightarrow[\nu]{\tau} U'[\ ]$ and $W[\ ] \xrightarrow[\tau]{\nu} W'[\ ]$ for some ν such that $F'[b] \equiv U'[W'[b]]$. As $W[b] \equiv \mu.V[b]$ the definition of $W[\ ] \xrightarrow[\tau]{\mu} W'[\ ]$ gives $\mu.V[A] \xrightarrow{\nu} W'[B]$ whenever $A \xrightarrow{\tau} B$, hence $\mu=\nu$ and $W'[B] \equiv V[A]$; as this holds whenever $A \xrightarrow{\tau} B$ it must be true that W' is constant and $W[\ ] \xrightarrow{\nu} W'[\ ]$. Then as $U[\ ] \xrightarrow[\nu]{\tau} U'[\ ]$ it follows from proposition 4.7 that $U[W[A]] \xrightarrow{\tau} U'[W'[A]]$ for all A, that is $G[A] \xrightarrow{\tau} F'[A]$, so we can conclude that $F[A] \overset{\xi}{\Rightarrow} F'[A]$ as required.

(f) $F[b] \equiv G[b] \mid C$.

In this case there are three possible derivations for $F[\ ] \xrightarrow[\tau]{} F'[\ ]$, firstly $G[\ ] \xrightarrow{\tau} G'[\ ]$ and $F'[b] \equiv G'[b] \mid C$, secondly $C \xrightarrow{\tau} D$ with G constant and $F'[b] \equiv G[b] \mid D$, and thirdly $F[\ ] \xrightarrow[\tau]{\lambda} F'[\ ]$ and $C \xrightarrow{\bar{\lambda}} D$ for some $\lambda \neq \Lambda$ with $F'[b] \equiv G'[b] \mid D$. The result follows trivially in the second case, and the other cases present no difficulty applying the structural inductive hypothesis to G.

(g) $F[b] \equiv G[b][S]$.

Here $F[\ ] \xrightarrow[\tau]{} F'[\ ]$ implies that $G[\ ] \xrightarrow[\tau]{} G'[\ ]$ and $F'[b] \equiv G'[b][S]$. The required result is obtained by applying the structural inductive hypothesis to G.

In all cases we have shown that the result holds completing the structural induction and hence the proof of the lemma.

Having proved this result we are ready to show that firmly-guarded contexts do satisfy condition (b) imposed in theorem 4.12, namely that such a context and all its possible derivatives preserves $\approx_k$ for all $k \geq 0$.

Theorem 4.19

If F is a firmly-guarded context then F and all its possible derivatives preserve $\approx_k$ for all $k \geq 0$.

Proof

As a consequence of the definition of a firmly-guarded context, it follows that any possible derivative of such a context is also firmly guarded; hence it is sufficient to prove that any firmly-guarded context preserves $\approx_k$. This will be done by induction on k, the result being trivial at k=0.

For the inductive step at k+1 assuming that all firmly-guarded contexts preserve $\approx_k$, we have to assume that $A_1 \approx_{k+1} A_2$ and prove that $F[A_1] \approx_{k+1} F[A_2]$. So assuming $F[A_1] \stackrel{t}{\Rightarrow} D$ it is necessary to show that $F[A_2] \stackrel{t}{\Rightarrow} E$ for some E such that $E \approx_k D$. Corollary 4.11 applied to $F[A_1] \stackrel{t}{\Rightarrow} D$ gives two possibilities:

(i) $F[\ ] \stackrel{t}{\underset{s}{\Rightarrow}} F'[\ ]$ for some $s \in Act^+ \cup \{\tau\}$, $A_1 \stackrel{s}{\Rightarrow} B_1$ and $D \equiv F'[B_1]$. In this case we have, as $A_1 \approx_{k+1} A_2$, $A_2 \stackrel{s}{\Rightarrow} B_2$ with $B_1 \approx_k B_2$. Then unless $s=\tau$, $A_2 \stackrel{s}{\Rightarrow} B_2$, and applying proposition 4.7 $F[A_2] \stackrel{t}{\Rightarrow} F'[B_2]$. Now as F' is firmly guarded it preserves $\approx_k$, hence $F'[B_2] \approx_k F'[B_1] \equiv D$ as required. In the case $s=\tau$ we have $A_2 \stackrel{\tau}{\Rightarrow} B_2$, and if $A_2 \stackrel{\tau}{\Rightarrow} B_2$ proposition 4.7 can be applied as above. Otherwise we need to use lemma 4.18: we have $F[\ ] \stackrel{t}{\underset{\tau}{\Rightarrow}} F'[\ ]$ so $F[\ ] \stackrel{t_0}{\Rightarrow} F_0[\ ] \underset{\tau}{\rightarrow} F_1[\ ] \stackrel{t_1}{\Rightarrow} F'[\ ]$ and $t=t_0 \hat{\ } t_1$, and as F_0 must be firmly guarded $F_0[\ ] \stackrel{\hat{\tau}}{\Rightarrow} F_1[\ ]$, hence $F[\ ] \stackrel{t}{\Rightarrow} F'[\ ]$. Then

$F[A_2] \stackrel{t}{\Rightarrow} F'[A_2]$, and as $A_2 \stackrel{t}{\Rightarrow} B_2$ but it is not the case that $A_2 \stackrel{\tau}{\Rightarrow} B_2$ it must follow that $A_2 \equiv B_2$, hence $F[A_2] \stackrel{t}{\Rightarrow} F'[B_2]$. As before the inductive hypothesis is used to show that $F'[B_2] \approx_k D$ to complete the proof.

(ii) $F[\ ] \stackrel{t}{\Rightarrow} F'[\ ]$ and $D \equiv F'[A_1]$. The proof is trivial in this case; we can deduce immediately that $F[A_2] \stackrel{t}{\Rightarrow} F'[A_2]$ and use the inductive hypothesis to show that $F'[A_2] \approx_k D$.

Having shown that $F[A_2] \stackrel{t}{\Rightarrow} E$ with $D \approx_k E$ whenever $F[A_1] \stackrel{t}{\Rightarrow} D$ we can repeat this proof with the rôles of A_1 and A_2 interchanged to complete the proof that $F[A_1] \approx_{k+1} F[A_2]$.

Theorems 4.17 and 4.19 have demonstrated that certain classes of contexts satisfy the requirements (a) and (b) respectively for theorem 4.12. We can combine these results to give a theorem that can be easily applied to proofs in CCS.

Theorem 4.20

Let F be a firmly-guarded context of the form

$$F[b] \equiv G[\lambda.C[b]]$$

where neither C nor G is of the form $H[B|D[b]]$ with $\bar{\lambda} \in L_B$ (where L_B is the sort of B) and assume that λ is not relabelled anywhere in F. Then F is a contracting transformation.

Proof

By theorem 4.17 F satisfies the first condition of theorem 4.12, and by theorem 4.19 F satisfies the second condition of theorem 4.12. Hence F is contracting.

As an example of the use of this theorem we consider a specification of a scheduling agent to control the access to a shared resource. The constraint to be imposed is that the number of occurrences of α-actions since the setting-up of the agent is always at least at large as the number of occurrences of β-actions (in a practical example this could be used to ensure that the number of items removed from a queue never exceeds the number of items put into the queue, thus preventing any attempt to read from an empty queue). This type of restriction can be expressed easily using path expressions [C+H]; in a program written in Path Pascal [K+C] the constraint would be imposed by a statement

path a; b end

where a and b are procedures that perform the α- and β-actions respectively. The specification in CCS may be written using a parameter to indicate the excess of occurrences of α-actions over β-actions:

$S(0) \Leftarrow \alpha.S(1)$
$S(i) \Leftarrow \alpha.S(i+1) + \beta.S(i-1)$ for $i \geq 1$

Suppose it is required to construct an agent which meets the specification of S(0) in pure CCS without using any parameterisation. One solution is the behaviour

$B \Leftarrow \alpha.(\beta.NIL|B)$

which effectively spurns a new process to perform a β-action after each occurence of a α-action; this process may be "activated" at any time in the future.

We wish to show that the agent B satisfies the specification S(0). A direct proof of this would appear to be extremely awkward due to the fact that the derivatives of B include agents such as

$\beta.NIL|\beta.NIL|....|\beta.NIL|B$

of arbitrary length. However, it can be observed from the definition of B given above that $B \approx F[B]$ where

$F[b] \equiv \alpha.(\beta.NIL|b)$.

As F satisfies the conditions of theorem 4.20 it is a contracting transformation; hence if it can be proved that $S(0) \approx F[S(0)]$ it can be concluded that $B \approx S(0)$, and hence, as both agents are stable, $B = S(0)$. Theorem 4.20 thus effectively allows us to show that the specification satisfies the construction as an alternative to the usual approach of showing that the constructed behaviour meets its specification.

To show that $S(0) \approx F[S(0)]$ we require an intermediate result that $\beta.NIL|S(0) \approx S(1)$, and can then proceed:

$$\begin{aligned} F[S(0)] &= \alpha.(\beta.NIL|S(0)) \\ &= \alpha.S(1) \text{ (by the intermediate result)} \\ &= S(0) \end{aligned}$$

It remains to show that $\beta.NIL|S(0) \approx S(1)$. As the specification of S(i) for $i \geq 1$ was not used in the argument above and obviously must be used for all i in any proof that B meets the specification it is clear that it must be required in this demonstration. A close examination of the behaviours should convince the reader that $\beta.NIL|S(i) \approx S(i+1)$ holds for all $i \geq 0$, but since the definition of S(i) uses both S(i-1) and S(i+1) it is not possible to prove this result directly by induction. In fact no technique so far introduced appears to be able to prove this required intermediate result; however applying the alternative definition of observational equiavalence to be introduced in the next chapter the proof becomes remarkably straightforward, so it is deferred until the appropriate theory has been introduced.

FURTHER DEVELOPMENTS

The study of contracting transformations has been limited to simple contexts (with just one "hole") and to the pure subcalculus of CCS. In order to apply any proof techniques using the contracting property to practical examples it is necessary for the research to be extended to the full calculus and also to more general transformations. One extension which should be extremely straightforward is to consider contexts of the form

$$F[b] \equiv G[\, H_1[b] + H_2[b] \,].$$

In view of the earlier remarks on mutual "swallowing" it is not unreasonable to postulate that such a context should be contracting if it is possible to show, by the techniques used in this chapter, that $G[H_1[\ ]]$ and $G[H_2[\ ]]$ are both contracting, using the same λ_0 in each case. It is not anticipated that the proof of such a result would present any difficulties using an approach almost identical to that taken in this chapter. It should be easy to treat more complex "sum" contexts in the same way.

The extension of the results of this chapter to the full calculus should not present too many difficulties, as long as certain limitations are imposed on the use of conditional expressions, probably requiring that if the label λ_0 used in defining the measure $\sharp$ occurs only within a conditional expression then it must occur within both "arms" of that conditional.

The treatment of contexts with more than one hole which are not sums of simple contexts, and of mutually recursive equations such as

$$b_1 \simeq F[b_1, b_2]$$
$$b_2 \simeq F[b_1, b_2]$$

will require a more general form of the "causing" relationship introduced in this chapter. It is not clear how much more complex this will have to be, although it seems likely that a proof of a result analogous to, say, proposition 4.15, if it were required, would have to be extremely complex, and it is clear that the work involved in extending the conclusions of this chapter will be quite lengthy if the same approach is used. However, such general results may be extremely beneficial to proof techniques in CCS.

Another interesting topic for future research would be to attempt to relate contracting transformations in CCS to contraction mappings on metric spaces [Sut]. A function F is said to be a contraction mapping if there exists some $K < 1$ such that (if d denotes the metric)

$$d(F[A],F[B]) < Kd(A,B)$$

for all A and B. A well-known result due to Banach states that in complete metric spaces such functions have unique fixed points.

To relate our contracting transformations to this general concept it will be necessary to define a distance function d between behaviour expressions such that $d(A,B) = 0$ if and only if $A \approx B$, a metric space thus being obtained by taking the quotient space (of equivalence classes under $\approx$). The question of an appropriate distance function presents interesting opportunities for research; the metric defined on trees by Arnold and Nivat [A+N] may provide a useful starting point for behaviours constructed from identifiers, NIL, |, + non-τ guarding, but difficulties may emerge in deciding how to treat τ guards and the restriction and relabelling operators. If an appropriate metric can be obtained it would be interesting to discover whether the conditions of theorem 4.20 ensure that a behaviour transformation is a contraction mapping, and to study product spaces in the hope of gaining a useful insight regarding the treatment of more general behaviour transformations discussed above.

Further comments on relating the work presented in this chapter to the alternative definition of observational equivalence studied in the next chapter and to recent research by Milner [Mil5] and by Hennessy and Plotkin [H+P] can be found in the concluding remarks at the end of this thesis.

5 BISIMULATION TECHNIQUES

INTRODUCTION

In this chapter an alternative definition of observational equivalence suggested by David Park is introduced, which enables proofs of equivalence of behaviours to be performed by showing the existence of bisimulations [Par]. An algorithm for the construction of bisimulations is presented with examples of its use. The new equivalence is defined to be the maximal fixed-point of the relation that is used to obtain $\approx_{k+1}$ from $\approx_k$, using the partial ordering of set inclusion. It will be shown that this alternative equivalence, which will be written as $\approx_F$ to denote that it is a fixed-point, is stronger than the original observational equivalence, $\approx$, and hence that in order to prove that two behaviours A and B are observationally equivalent it is sufficient to demonstrate that $A \approx_F B$.

It will be shown that a simpler defining relation than the one used to obtain $\approx_{k+1}$ from $\approx_k$ (using derivation strings of length 0 or 1 only) gives the same maximal fixed-point $\approx_F$, simplifying proofs that $A \approx_F B$. Furthermore, using a standard and elementary result from fixed-point theory, it will be shown that to prove that $A \approx_F B$ it is adequate to demonstrate the existence of a relation R such that $\langle A,B \rangle \in R$ and $R \subseteq E(R)$, where E is the aforementioned simplified defining relation. As E involves only single-length derivations the construction of such a relation, or bisimulation, is extremely straightforward in many cases, and although the process may seem somewhat lengthy at times, an algorithm will be displayed (for the pure calculus), which subject to certain restrictions on the behaviours A and B, guarantees to construct a bisimulation or demonstrate that one does not exist, providing there are only finitely many possible derivatives of A and B to examine. It appears likely that this algorithm could be mechanised without too much difficulty, especially if restrictions

on behaviours could be found which would ensure its termination; so the fact that the construction could be somewhat lengthy is not unduly worrying. An example will be given to show how the algorithm can be applied to the full calculus with value-passing, followed by a discussion of the necessary adaptations.

Milner chose to use the new definition of equivalence for calculi similar to CCS in [Mil5]; it is likely that in future work in related areas it will be found more appropriate than the original approach of [Mil4].

MAXIMAL FIXED-POINTS

For convenience of notation and readability we shall work in the pure CCS without value-passing and parameterised behaviours for much of this chapter, and subsequently discuss how the results presented can be adapted to the full calculus.

It will be recalled, from [Mil4] and the second chapter of this thesis that $\approx$ is defined as

$$\approx \;=\; \bigcap_{k=0}^{\infty} \approx_k,$$

where $\approx_k$ is defined for $k \geq 0$ as a decreasing sequence of relations by:

$A \approx_0 B$ for all behaviours A and B;

$A \approx_{k+1} B$ <u>if and only if</u> for all $s \in \Lambda^*$,

$A \stackrel{s}{\Rightarrow} A'$ implies that $B \stackrel{s}{\Rightarrow} B'$ for some B' such that $A' \approx_k B'$

<u>and</u> $B \stackrel{s}{\Rightarrow} B'$ implies that $A \stackrel{s}{\Rightarrow} A'$ for some A' such that $A' \approx_k B'$.

The latter half of the above definition can be expressed in the form

$$\approx_{k+1} = E'(\approx_k),$$

where, expressing relations in the form of sets of ordered pairs, E' is given by the following definition:

Definition

$$E'(R) = \{ \langle x,y\rangle : x \stackrel{s}{\Rightarrow} x' \supset \exists y'.(y \stackrel{s}{\Rightarrow} y' \wedge \langle x',y'\rangle \in R) \wedge y \stackrel{s}{\Rightarrow} y' \supset \exists x'.(x \stackrel{s}{\Rightarrow} x' \wedge \langle x',y'\rangle \in R) \},$$
for $s \in \Lambda^*$.

E' was used rather than E to denote this defining relation as it will be shown later that a simpler defining relation gives the same maximal fixed-point and as this relation will be used subsequently it will be called E. It should be observed that $\approx$ is not a fixed point of E', that is it is not the case that $\approx = E'(\approx)$. To demonstrate this it is necessary to show two behaviours A and B such that $A \approx B$ but $\langle A,B\rangle \notin E'(\approx)$. No simple example is known; we outline one constructed by Milner but unpublished. Behaviours u_i and v_i for $i \geq 0$ are defined inductively by

$u_0 \Leftarrow \beta.NIL,$
$v_0 \Leftarrow \gamma.NIL,$
$u_{i+1} \Leftarrow \alpha.(u_i + v_i)$ and
$v_{i+1} \Leftarrow \alpha.u_i + \alpha.v_i.$

It can be shown without too much difficulty that $u_i \approx_i v_i$ but $u_i \not\approx_{i+1} v_i$ and $u_i \not\approx_{i+1} u_i + v_i \not\approx_{i+1} v_i$ for $i \geq 0$. Infinite behaviours U_k and V_k are then defined by

$U_k \Leftarrow u_k + \tau.U_{k+1}$ and
$V_k \Leftarrow v_k + \tau.V_{k+1},$

and $W_{k,L}$ is defined as V_k for $L=0$ and for $L>0$ by

$$W_{k,L} \not\approx U_k + \tau.W_{k+1,L-1}$$

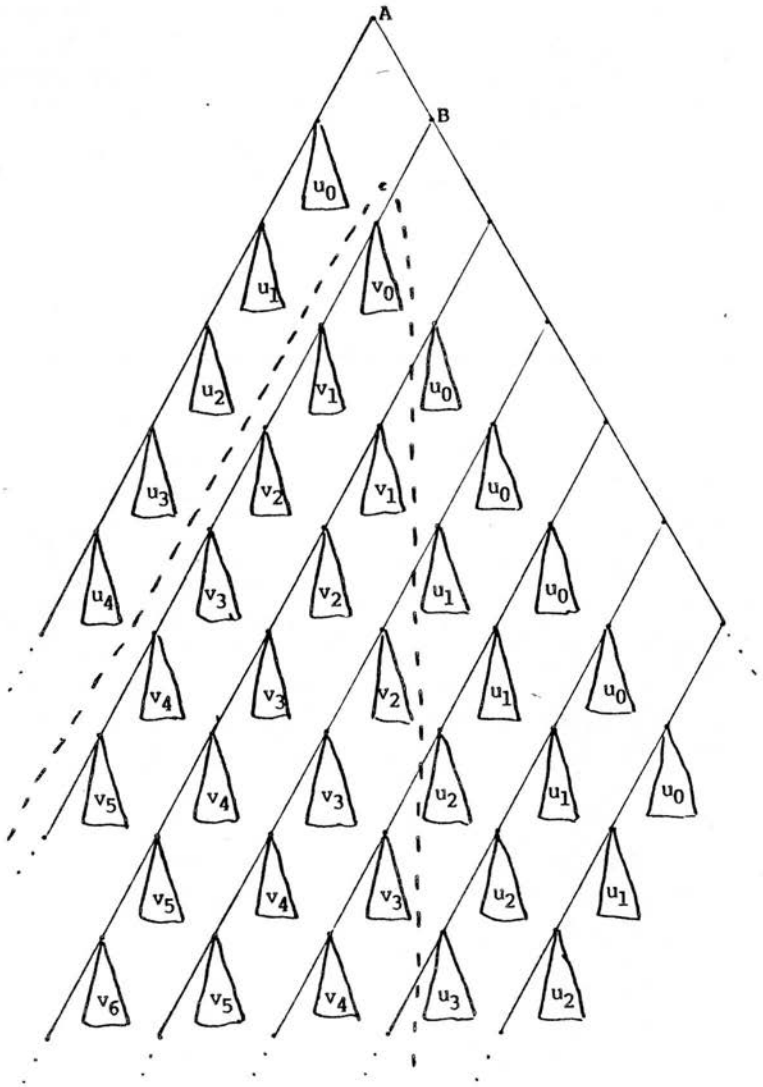
It can be shown by induction on 1 that $U_k \approx_{k+L} W_{k,L}$ and it can also be seen that $U_k \not\approx_{k+L+1} W_{k,L}$. Finally, behaviours A and B are defined by

$$B = \tau.W_{0,0} + \tau.(\tau.W_{0,1} + \tau.(.....)) \text{ and}$$
$$A = \tau.U_0 + \tau.B.$$

The structure of the agents A and B is illustrated in the diagram on the next page; the area surrounded by the broken line can be seen to contain agents V_i, with U_i occurring outside this line. It is not hard to check that $A \approx B$. However $A \stackrel{\varepsilon}{\Rightarrow} U_1$ (as $U_0 \stackrel{\varepsilon}{\Rightarrow} U_1$) so if $\langle A,B\rangle \bullet E'(\approx)$ there would be some Z such that $B \stackrel{\varepsilon}{\Rightarrow} Z$ and $Z \approx U_1$. As U_1 can undergo an $\alpha\beta$-derivation but will not accept a β- or δ-experiment the only possible hopes for Z are seen to be $W_{1,k}$ for $k \geq 0$. However $U_1 \not\approx_{2+k} W_{1,k}$ so there is no such Z and $\langle A,B\rangle \notin E'(\approx)$.

David Park suggested that it might be beneficial to study an alternative definition of observational equivalence, considering the maximal fixed-point of the defining relation E', using the partial ordering of set inclusion. There follows an introduction to the properties of this definition, and the associated concept of a bisimulation; the results 5.2 to 5.5 were postulated by Park and Milner, and their proofs are entirely standard. Subsequently we introduce results enabling the easy construction and checking of bisimulations, leading to the aforementioned algorithm. In order to use Park's definition it is necessary to know that a maximal fixed point does exist; this is a standard result from the theory of partial orderings, originally due to Tarski [Tar]:

An example to show that $\alpha \notin E'(\alpha)$



Proposition 5.1

If a function F is monotonic over a partial ordering and y is defined by

$$y = \bigsqcup \{ x : x \sqsubseteq F(x) \}$$

then y is the maximal fixed point of F.

Proof

It is necessary to show (i) that y is a fixed point of F, and (ii) that it is maximal, that is if $z = f(z)$ then $z \sqsubseteq y$.

(i) Suppose $x \sqsubseteq F(x)$ then, from the definition of y, $x \sqsubseteq y$, and as F is monotonic $F(x) \sqsubseteq F(y)$, so $x \sqsubseteq F(y)$. As this holds for all x such that $x \sqsubseteq F(x)$ it follows that $\sqcup \{ x : x \sqsubseteq F(x) \} \sqsubseteq F(y)$, that is $y \sqsubseteq F(y)$. Additionally, it follows from the monotonicity of F that $F(y) \sqsubseteq F(F(y))$, so $F(y) \in \{ x : x \sqsubseteq F(x) \}$, hence $F(y) \sqsubseteq \sqcup \{ x : x \sqsubseteq F(x) \} = y$, completing the proof that $y = F(y)$.

(ii) If $z = F(z)$ it follows trivially that $z \sqsubseteq F(z)$, hence $z \in \{ x : x \sqsubseteq F(x) \}$, so $z \sqsubseteq \sqcup \{ x : x \sqsubseteq F(x) \} = y$ as required.

It has been proved that y is a fixed point of F and that it is greater than any other fixed-point, so y is the maximal fixed-point of F as required.

In the partial ordering of set-inclusion the monotonicity requirement on F is simply that if $A \subseteq B$ then $F(A) \subseteq F(B)$ and it is easy to see that E' is monotonic simply by considering the definition. Hence we define $\approx_F$, the maximal fixed-point

equivalence, as follows.

Definition

$$\approx_F = \cup\{ R : R \subseteq E'(R) \}.$$

It should be noted that we have not yet proved that $\approx_F$ is an equivalence relation. Although this is strictly unnecessary for the purposes of being able to use $\approx_F$ in proving observational equivalence, we need to demonstrate that it is an equivalence relation in order to justify the terminology used in this chapter.

Proposition 5.2

$\approx_F$ is an equivalence relation.

Proof

We have to demonstrate that $A \approx_F A$ for all agents A, that $B \approx_F A$ whenever $A \approx_F B$ and that if $A \approx_F B$ and $B \approx_F C$ then $A \approx_F C$.

The proof of reflexivity is straightforward: given A, to show that $A \approx_F A$ we define

$$R = \{ \langle x,x \rangle : x \text{ is a derivative of } A \}$$

then it is easy to see that $\langle A,A \rangle \in R$ and $R \subseteq E'(R)$, so $A \approx_F A$ as required.

To prove that $A \approx_F B$ implies $B \approx_F A$ we know that $\langle A,B \rangle \in \cup\{ R : R \subseteq E'(R) \}$, hence $\langle A,B \rangle \in R$ for some relation R such that $R \subseteq E'(R)$. Then defining S by

$S = \{ \langle y,x \rangle : \langle x,y \rangle \in R \}$

we have $\langle B,A \rangle \in S$ and $S \subseteq E'(S)$, hence $B \approx_F A$ as required.

Finally to prove transitivity we have to show that $A \approx_F B$ and $B \approx_F C$ together imply that $A \approx_F C$. We know that $\langle A,B \rangle \in R$ for some R such that $R \subseteq E(R)$ and $\langle B,C \rangle \in S$ for some S such that $S \subseteq E'(S)$, hence we define T by

$T = \{ \langle x,z \rangle : \langle x,y \rangle \in R$ and $\langle y,z \rangle \in S$ for some $y \}$

so that $\langle A,C \rangle \in T$. To show that $T \subseteq E'(T)$, we suppose that $\langle x,z \rangle \in T$, and have to show that $\langle x,z \rangle \in E'(T)$. So given $x \stackrel{s}{\Rightarrow} x'$ we have to show that $z \stackrel{s}{\Rightarrow} z'$ with $\langle x',z' \rangle \in T$. We know that for some y, $\langle x,y \rangle \in R$ and $\langle y,z \rangle \in S$, hence $\langle x,y \rangle \in E'(R)$ and $\langle y,z \rangle \in E'(Z)$. So $x \stackrel{s}{\Rightarrow} x'$ implies that $y \stackrel{s}{\Rightarrow} y'$ with $\langle x',y' \rangle \in R$, hence $z \stackrel{s}{\Rightarrow} z'$ with $\langle y',z' \rangle \in S$, so $\langle x',z' \rangle \in T$ as required, completing the proof of transitivity, and hence showing that $\approx_F$ is an equivalence relation.

In order to use $\approx_F$ to prove observational equivalence of behaviours we need to show that it is a stronger equivalence than $\approx$.

Proposition 5.3

If $A \approx_F B$, then $A \approx B$.

Proof

We assume that $R \subseteq E'(R)$ and prove that $R \subseteq \approx$ by using induction on k to show that $R \subseteq \approx_k$ for all $k \geq 0$. The base step with $k=0$ is trivial; we consider the inductive step at $k+1$.

By the inductive hypothesis we have $R \subseteq \simeq_k$. As E' is monotonic it follows that $E'(R) \subseteq E'(\simeq_k) = \simeq_{k+1}$, and as $R \subseteq E'(R)$ we can conclude that $R \subseteq \simeq_{k+1}$ as required.

Having completed the inductive step we have shown that $R \subseteq \simeq$ whenever $R \subseteq E'(R)$; hence $\cup\{ R : R \subseteq E'(R) \} \subseteq \simeq$, that is $\simeq_F \subseteq \simeq$, completing the proof.

By this proposition in order to prove that $A \simeq B$ it is enough to show that $A \simeq_F B$, and to do this it is adequate to demonstrate the existence of some relation R with $\langle A,B \rangle \in R$ such that $R \subseteq E'(R)$. This involves investigating derivations of behaviours by strings $s \in A^*$. It would be much more convenient to be able to work with derivation strings of unit length; hence we introduce a simpler defining relation, E, and prove that this gives the same maximal fixed-point as E'.

Definition

$$E(R) = \{ \langle x,y \rangle : x \xrightarrow{s} x' \Rightarrow \exists y'.(y \xrightarrow{s} y' \wedge \langle x',y' \rangle \in R) \wedge y \xrightarrow{s} y' \Rightarrow \exists x'.(x \xrightarrow{s} x' \wedge \langle x',y' \rangle \in R) \},$$

for $s \in A \cup \{\varepsilon\}$.

To be able to use this relation E in place of E' we need to show that it gives rise to the same maximal fixed-point, $\simeq_F$, that is we have to prove that $\simeq_F = \cup\{ R : R \subseteq E(R) \}$. To do this we begin by demonstrating that $R \subseteq E(R)$ if and only if $R \subseteq E'(R)$.

Proposition 5.4

With E and E' defined as above, for any set R of ordered pairs, $R \subseteq E(R)$ if and only if $R \subseteq E'(R)$.

Proof

The proof that $R \subseteq E'(R)$ implies $R \subseteq E(R)$ is trivial as it follows from the definitions that, for any relation R, $E'(R) \subseteq E(R)$.

To show that $R \subseteq E(R)$ implies $R \subseteq E'(R)$ we assume that $\langle x,y\rangle \in R \subseteq E(R)$ and have to show that $\langle x,y\rangle \in E'(R)$. To do this we need to show that if $x \stackrel{s}{\Rightarrow} x'$ for $s \in \Lambda^*$ then $y \stackrel{s}{\Rightarrow} y'$ for some y' such that $\langle x',y'\rangle \in R$ (and vice versa). So suppose $x \stackrel{s}{\Rightarrow} x'$. Then expressing the string s in the form $s = s_1 \ldots s_n$ with each $s_i \in \Lambda \cup \{\tau\}$, we have $x \stackrel{s_1 \ldots s_n}{\Longrightarrow} x'$, hence there exists a sequence of agents x_i for $i \in \{0,\ldots,n\}$ such that $x_0 = x$, $x_n = x'$ and $x_{i-1} \stackrel{s_i}{\Rightarrow} x_i$ for $i \in \{1,\ldots,n\}$. Using an inductive technique we construct a sequence y_i for $i \in \{0,\ldots,n\}$ such that $y_0 = y$, $y_{i-1} \stackrel{s_i}{\Rightarrow} y_i$ for $i \in \{1,\ldots,n\}$ and $\langle x_i,y_i\rangle \in R$ for $i \in \{0,\ldots,n\}$:

(i) For the base we define $y_0 = y$, and the conditions are satisfied as $\langle x,y\rangle \in R$ by assumption.

(ii) To obtain y_{i+1} from y_i we use the inductive assumption that $\langle x_i,y_i\rangle \in R$, so as $R \subseteq E(R)$ we can deduce that $\langle x_i,y_i\rangle \in E(R)$, and as $x_i \stackrel{s_{i+1}}{\Longrightarrow} x_{i+1}$ it follows that $y_i \stackrel{s_{i+1}}{\Longrightarrow} z$ for some z such that $\langle x_{i+1},z\rangle \in R$. Defining y_{i+1} to be equal to z meets the requirements for y_{i+1}.

Having constructed the sequence y_i for $i \in \{1,\ldots,n\}$ we define y' to be y_n, then we have $y \stackrel{s}{\Rightarrow} y'$ and $\langle x',y'\rangle \in R$. This can be done whenever $x \stackrel{s}{\Rightarrow} x'$ for $s \in \Lambda^*$, and we can do the same thing interchanging the rôles of x and y hence completing the proof that $\langle x,y\rangle \in E'(R)$, and thus demonstrating that $R \subseteq E(R)$ implies $R \subseteq E'(R)$.

Corollary 5.5

E and E' have the same maximal fixed point, $\approx_F$.

Proof

Define $\approx_G = \cup\{ R : R \subseteq E(R) \}$, then by proposition 5.1 $\approx_G$ is the maximal fixed point of E. As $E'(R) \subseteq E(R)$ it follows trivially that $\approx_F \subseteq \approx_G$. Conversely, $\approx_G \subseteq E'(\approx_G)$ by proposition 5.4 so $\approx_G \subseteq \{ R : R \subseteq E'(R) \}$, that is $\approx_G \subseteq \approx_F$. Hence $\approx_F = \approx_G$ as required, completing the proof.

Having shown that $\approx_F = \cup\{ R : R \subseteq E(R) \}$, in order to prove that $A \approx_F B$ for two agents A and B (and hence, by proposition 5.3, that $A \approx B$), it is sufficient to demonstrate the existence of a relation R such that $\langle A,B \rangle \in R$ and $R \subseteq E(R)$. This often turns out to be routine and straightforward - in many cases such an R can be written down without any work. As an example we consider a simplified version of a scheduler introduced in chapter 10 of [Mil4]. We define:

$s \Leftarrow (\alpha|\beta).s$
$c \Leftarrow \alpha.\gamma.c$
$d \Leftarrow \beta.\bar{\gamma}.d$

and wish to show that $(c|d)\backslash\gamma = s$. (The notation used in the definition of s is the composite guarding introduced in chapter 3). As both expressions are stable it will be enough to prove observational equivalence, hence all we need do is show the existence of a relation R such that $\langle s,(c|d)\backslash\gamma \rangle \in R$ and $R \subseteq E(R)$. It is fairly clear that the following relation satisfies these requirements:

$R = \{ \langle s,(c,d)\backslash\gamma \rangle,$
$\langle \beta.s,(\gamma.c|d)\backslash\gamma \rangle,$

$\langle \alpha.s, (c|\bar{\alpha}.d)\backslash\alpha \rangle$,
$\langle s, (\alpha.c|\bar{\alpha}.d)\backslash\alpha \rangle$ }.

The proof that $R \leq E(R)$ is not too difficult; it will be made even easier by the next proposition.

Following Park in [Par] we shall refer to a relation R such that $R \leq E(R)$ and $\langle A,B \rangle \in R$ as a bisimulation between A and B. We now present a more theoretical example of the construction of such a relation.

We wish to show that $A \simeq_F B$ implies that $A|D \simeq_F B|D$ for any behaviour D and to do this we define a relation S by

$$S = \{ \langle x|z, y|z \rangle : x \simeq_F y,\ z \text{ any behaviour} \}$$

and prove that $S \leq E(S)$, then as it is trivially true that $\langle A|D, B|D \rangle \in S$ it follows that $A|D \simeq_F B|D$. As all pairs in the relation S are of the form $\langle x|z, y|z \rangle$ to prove that $S \leq E(S)$ it is necessary to demonstrate that $\langle x|z, y|z \rangle \in E(S)$ whenever $\langle x,y \rangle \in S$. To do this we have to show that whenever $x|z \overset{s}{\Rightarrow} u$ (for $s \in \Lambda \cup \{\varepsilon\}$) there exists some v such that $y|z \overset{s}{\Rightarrow} v$ and $\langle u,v \rangle \in S$ (and vice versa). So suppose $x|z \overset{s}{\Rightarrow} u$; there are two possible derivation sequences, either $x \overset{r \leq t}{\Longrightarrow} x'$ and $z \overset{\bar{r} \leq \bar{t}}{\Longrightarrow} z'$ or $x \overset{r t}{\Longrightarrow} x'$ and $z \overset{\bar{r} \leq \bar{t}}{\Longrightarrow} z'$, with $u = x'|z'$ in either case. As $x \simeq_F y$ there exists some y' such that $x' \simeq_F y'$ and either $y \overset{r \leq t}{\Longrightarrow} y'$ (in the first case) or $y \overset{r t}{\Rightarrow} y'$ (in the second). In either case $y|z \overset{s}{\Rightarrow} v$ where $v = y'|z'$ and $\langle u,v \rangle \in S$ as required.

We now wish to develop a systematic method for the construction of a bisimulation R such that $\langle A,B \rangle \in R$ and $R \leq E(R)$ given behaviours A and B. It is found that it is much more convenient to be able to work with derivations of the form $\overset{\lambda}{\rightarrow}$ and $\overset{\tau}{\rightarrow}$ rather than $\overset{s}{\Rightarrow}$, hence we present a result which allows us to restrict ourselves to such derivations under certain conditions of stability and non-divergence.

Proposition 5.6

If a set R of ordered pairs of behaviours is such that $\langle x,y\rangle \in R$ implies that:

(i) x is stable;

(ii) y cannot diverge (that is, y cannot undergo an infinite sequence of τ-derivations;

(iii) if $y \xrightarrow{\tau} y'$ then $\langle x,y'\rangle \in R$;

(iv) if $y \xrightarrow{\lambda} y'$ there exists some behaviour x' such that $x \xrightarrow{\lambda} x'$ and $\langle x',y'\rangle \in R$; and

(v) if $x \xrightarrow{\lambda} x'$ and y is stable then there exists some behaviour y' such that $y \xrightarrow{\lambda} y'$ and $\langle x',y'\rangle \in R$

then $R \subseteq E(R)$.

Proof

It is necessary to show that $\langle x,y\rangle \in R$ implies that $\langle x,y\rangle \in E(R)$. To do this we assume $\langle x,y\rangle \in R$ and have to prove that, for $s \in A \cup \{\varepsilon\}$, (a) if $x \stackrel{s}{\Rightarrow} x'$ then there exists y' such that $y \stackrel{s}{\Rightarrow} y'$ and $\langle x',y'\rangle \in R$, and (b) if $y \stackrel{s}{\Rightarrow} y'$ then there exists x' such that $x \stackrel{s}{\Rightarrow} x'$ and $\langle x',y'\rangle \in R$.

(a) $x \stackrel{s}{\Rightarrow} x'$. In the case $s = \varepsilon$ the stability of x implies that $x' = x$, hence defining $y' = y$ we obtain $y \stackrel{\varepsilon}{\Rightarrow} y'$ and $\langle x',y'\rangle \in R$ as required. Otherwise $s = \lambda \in A$ and as x is stable we have $x \xrightarrow{\lambda} x_1 \stackrel{\varepsilon}{\Rightarrow} x'$. By assumption (ii) y cannot diverge so $y \xrightarrow{\tau^n} y_0$ for some stable behaviour y_0, and assumption (iii) implies that $\langle x,y_0\rangle \in R$. Then assumption (v) gives y' such that $y_0 \xrightarrow{\lambda} y'$ and $\langle x_1,y'\rangle \in R$. This implies (using assumption (i)) that x_1 is stable, and as $x_1 \stackrel{\varepsilon}{\Rightarrow} x'$ we conclude that $x' = x_1$, so $\langle x',y'\rangle \in R$ as required.

(b) $y \stackrel{s}{\Rightarrow} y'$. In the case $s = \varepsilon$ assumption (iii) implies

that $\langle x,y' \rangle \in R$, so defining $x' = x$ we have $x \stackrel{\varepsilon}{\Rightarrow} x'$ and $\langle x',y' \rangle \in R$ as required. Otherwise $s = \lambda \in \Lambda$ and we can split the derivation $y \stackrel{s}{\Rightarrow} y'$ into $y \stackrel{\varepsilon}{\Rightarrow} y_0 \stackrel{\lambda}{\rightarrow} y_1 \stackrel{\varepsilon}{\Rightarrow} y'$ for some behaviours y_0 and y_1. Assumption (iii) gives $\langle x,y_0 \rangle \in R$, so applying assumption (iv) we obtain x' such that $x \stackrel{\lambda}{\rightarrow} x'$ and $\langle x',y_1 \rangle \in R$, and hence using assumption (iii) again it follows that $\langle x',y' \rangle \in R$ as required.

This completes the proof that $\langle x,y \rangle \in E(R)$, hence $R \subseteq E(R)$.

This proposition will prove useful in the construction of bisimulations to prove the equivalence of behaviours. The stability condition is not unrealistic as proofs usually involve showing that a constructed agent is observationally equivalent to a rigid specification and all derivatives of a rigid behaviour are stable. If a behaviour is to meet its specification it is unlikely that it can be allowed to diverge, hence the second condition is also a reasonable one to impose.

To show that $R \subseteq E(R)$ in the scheduler example introduced earlier it is now sufficient to check that each of the five conditions in the statement of proposition 5.6 is satisfied by each ordered pair in the relation; there are only four pairs so this requires very little effort.

Returning to the example left unfinished in the previous chapter, it may be recalled that:

$S(0) \Leftarrow \alpha.S(1)$

$S(i) \Leftarrow \alpha.S(i+1) + \beta.S(i-1)$ for $i \geq 1$

It was required to prove that $\beta.NIL|S(0) \approx S(1)$. To do this we define the set R by

$R = \{ \langle \beta.NIL|S(i), S(i+1)\rangle, \langle NIL|S(i), S(i)\rangle : i \geq 0\}$.

As all the behaviours presented are rigid the first three conditions for proposition 5.6 hold trivially for all $\langle x,y\rangle \in R$, and very little work is necessary to check that the two remaining conditions hold. It may be concluded that $R \subseteq E(R)$, and hence R is a bisimulation between $\beta.NIL|S(0)$ and $S(1)$, proving their observational equivalence.

Proposition 5.6 presented a set of rules for checking that a relation is a bisimulation using only single derivations $\xrightarrow{\lambda}$ and $\xrightarrow{\tau}$. We now prove a converse result which holds if the agent A is rigid and is such that if $A \overset{s}{\Rightarrow} A_1$ and $A \overset{s}{\Rightarrow} A_2$ (for $s \in \Lambda^*$) then $A_1 \equiv A_2$. Once we have proved this it will be possible to use these conditions as a criterion for the fixed-point equivalence of behaviours which satisfy the constraints imposed in proposition 5.6 and this new condition, which shall be called total determinacy.

Definition

A behaviour A is said to be totally determinate if whenever $A \overset{\sigma}{\rightarrow} B$ and $A \overset{\sigma}{\rightarrow} C$ for some $\sigma \in (\Lambda \cup \{\tau\})^*$ then $B \equiv C$.

It should be noted that as we are working with rigid behaviours it is sufficient to consider $s \in \Lambda^*$ instead of $\sigma \in (\Lambda \cup \{\tau\})^*$

Proposition 5.7

If a behaviour A is rigid and totally determinate and there exists a bisimulation R between A and another behaviour B then there exists a set of ordered pairs S with $\langle A,B\rangle \in S$ such that for any $\langle x,y\rangle \in S$ the following conditions

hold:

(i) if $y \xrightarrow{\tau} y'$ then $\langle x,y' \rangle \in S$;

(ii) if $y \xrightarrow{\lambda} y'$ then there exists some behaviour x' such that $x \xrightarrow{\lambda} x'$ and $\langle x',y' \rangle \in S$; and

(iii) if $x \xrightarrow{\lambda} x'$ and y is stable then there exists some behaviour y' such that $y \xrightarrow{\lambda} y'$ and $\langle x',y' \rangle \in S$.

Proof

Define S by

$$S = \{ \langle x,y \rangle : \langle x,y \rangle \in R,\ x \text{ is a derivative of } A,\ y \text{ is a derivative of } B \}$$

then $S \subseteq R$ and $\langle A,B \rangle \in S$. We have to show that any pair $\langle x,y \rangle$ in S satisfies the three conditions.

(i) Suppose $y \xrightarrow{\tau} y'$; then as $\langle x,y \rangle \in R \subseteq E(R)$ it follows that $x \overset{\varepsilon}{\Rightarrow} x'$ for some behaviour x' such that $\langle x',y' \rangle \in R$. As A is rigid and x is a derivative of A it follows that x is stable so $x' = x$, hence $\langle x,y' \rangle \in R$. As $\langle x,y \rangle \in S$ we know that x is a derivative of A and y is a derivative of B, hence y' is a derivative of B, so $\langle x,y' \rangle \in S$ as required.

(ii) Suppose $y \xrightarrow{\lambda} y'$; then as $\langle x,y \rangle \in R \subseteq E(R)$ there exists some behaviour x' such that $x \overset{\lambda}{\Rightarrow} x'$ and $\langle x',y' \rangle \in R$. As x is a derivative of A which is rigid it follows that $x \xrightarrow{\lambda} x'$. Additionally, as x is a derivative of A and y is a derivative of B it follows that x' is a derivative of A and y' is a derivative of B, so $\langle x',y' \rangle \in S$ as required.

(iii) Suppose y is stable and $x \xrightarrow{\lambda} x'$; then as $\langle x,y \rangle \in R \subseteq E(R)$ it follows that $y \overset{\lambda}{\Rightarrow} y_1$ for some y_1 such that

$\langle x', y_1 \rangle \in R$. Then as y is stable it follows that $y \xrightarrow{\lambda} y' \xrightarrow{\varepsilon} y_1$ for some behaviour y'. By (ii) above this implies that there exists some behaviour x_1 such that $x \xrightarrow{\lambda} x_1$ and $\langle x_1, y' \rangle \in R$. As x is a derivative of A we have $A \xrightarrow{s} x$ for some $s \in A^*$, hence $A \xrightarrow{s\lambda} x'$ and $A \xrightarrow{s\lambda} x_1$. Then as A is rigid and totally determinate it follows that $x' \equiv x_1$ so $\langle x', y' \rangle \in R$. As x and y are derivatives of A and B respectively it follows that x' and y' are, hence $\langle x', y' \rangle \in S$ as required.

This completes the proof that any ordered pair in S satisfies the conditions (i) - (iii).

AN ALGORITHM TO CONSTRUCT A BISIMULATION

We present an algorithm which for a rigid behaviour A and another behaviour B which has no derivatives that can diverge will construct a set R satifying the conditions (iii) - (v) of proposition 5.6 if such a set exists, as long as A and B have only finitely many different derivatives. By proposition 5.6 such a set will be a bisimulation. If the algorithm terminates in failure and the behaviour A is also totally determinate then it can be concluded from proposition 5.7 that no bisimulation exists, and hence that the behaviours are not equivalent under the fixed-point definition.

The algorithm is essentially one of traversal of the derivation tree of the agent A, at each node defining an equivalence with a derivative of B. Failure occurs if at any point it is impossible to make define a valid equivalence, however if the behaviour A is not totally determinate some choice of equivalences may have been made at an earlier step, hence it is necessary to backtrack rather than failing immediately.

A bisimulation R between behaviours A and B is constructed by starting with just the ordered pair $\langle A,B \rangle$ in R and whenever a new ordered pair is put into the set this is checked against conditions (iii) - (v) of proposition 5.6, inserting new pairs into R as appropriate. Formally:

(i) Start with an empty set R.

(ii) Put the pair $\langle A,B \rangle$ into R and mark it as unchecked.

(iii) Check each pair $\langle x,y \rangle$ inserted into R by the following procedure:

(a) For each y' such that $y \xrightarrow{\tau} y'$ if $\langle x,y' \rangle$ is not already in R insert it into the set and mark it as unchecked.

(b) For each y' such that $y \xrightarrow{\lambda} y'$, if there is no $\langle x',y' \rangle$ yet in R such that $x \xrightarrow{\lambda} x'$, choose some x' such that $x \xrightarrow{\lambda} x'$ and put $\langle x',y' \rangle$ into R marking it as unchecked. If A is totally determinate there will only be one such x' hence no choice is necessary. If a choice has to be made it should be intuitively obvious when using the algorithm by hand which x' is appropriate; if the algorithm is being used mechanically the choice will have to be made systematically and recorded so that if the algorithm subsequently fails it is possible to backtrack and replace the chosen x' with another possibility. If no appropriate x' exists at this step the algorithm fails unless backtracking is possible.

(c) If y is stable it is necessary to check at this stage that for each x' such that $x \xrightarrow{\lambda} x'$ there is some pair $\langle x',y' \rangle$ in R such that $y \xrightarrow{\lambda} y'$. If not such a pair should be added

if possible (in the case where the behaviour A is totally determinate such a pair will already be in the set as a result of step (b) above if it exists); if this cannot be done then the algorithm fails unless it is possible to backtrack and alter some previously-made choice.

(d) If steps (a) - (c) have been completed succesfully then the pair $\langle x,y \rangle$ has been checked, hence the "unchecked" label should be removed from this pair.

(iv) Each new pair added to R should be checked in turn using the procedure in (iii) above; if all pairs have been checked succesfully the set R will satisfy the conditions of proposition 5.6 and will hence be a bisimulation.

If the number of possible derivatives of the behaviours A and B is finite then the number of pairs that have to be checked in applying the algorithm will also be finite, and hence the algorithm must terminate succesfully unless it fails at some stage.

Finitely-defined behaviours do not necessarily have only finitely many different derivatives; consider for example $A \Leftarrow \alpha.(A|A)$. The question of what restrictions need to be placed on behaviour definitions in order to ensure that the set of derivatives is finite, and hence to guarantee termination of the algorithm, is an area for future investigation. It may be that the exclusion of the composition operator from occurring within recursion would be a sufficient condition, but it is not clear whether any additional limitations on restriction and relabelling would be necessary. A possible approach would be to allow initially only prefixing and summation to occur within recursion (when it is easy to show that there are only finitely many

different derivatives if working in the pure calculus) and investigate what extensions can be made.

BISIMULATIONS IN THE FULL CCS

So far all the definitions and results in this chapter have been confined to the pure CCS without considering value-passing and parameterised behaviours. It is not difficult to see that all the results presented are valid for the full calculus; it remains only to consider how the algorithm introduced above needs to be adapted. We discuss this informally rather than expressing the algorithm again, but to introduce the concepts involved we first demonstrate the use of the algorithm for a non-trivial example.

The example we will consider comes from a set of lecture notes used by Milner to accompany a course on CCS and involves the representation of a workshop comprising two men, a mallet and a hammer. A job can be done by one man; if it is "easy" he need use no tools, a "hard" job requires the hammer, and other jobs can be done with either the hammer or the mallet. The behaviour of a man is:

$$
\begin{aligned}
&\mathrm{MAN} \Leftarrow \mathrm{IN}\ \mathrm{job}\ .\ \mathrm{READY}(\mathrm{job})\\
&\mathrm{READY}(\mathrm{job}) \Leftarrow \underline{\mathrm{if}}\ \mathrm{easy}(\mathrm{job})\ \underline{\mathrm{then}}\ \mathrm{D}(\mathrm{job})\\
&\qquad\qquad \underline{\mathrm{else\ if}}\ \mathrm{hard}(\mathrm{job})\ \underline{\mathrm{then}}\ \mathrm{GH}\ .\ \mathrm{PH}\ .\ \mathrm{D}(\mathrm{job})\\
&\qquad\qquad \underline{\mathrm{else}}\ \mathrm{GH}\ .\ \mathrm{PH}\ .\ \mathrm{D}(\mathrm{job}) + \mathrm{GM}\ .\ \mathrm{PM}\ .\ \mathrm{D}(\mathrm{job})\\
&\mathrm{D}(\mathrm{job}) \Leftarrow \overline{\mathrm{OUT}}\ \mathrm{job}\ .\ \mathrm{MAN}
\end{aligned}
$$

GH and PH represent "gethammer" and "puthammer", likewise GM and PM for the mallet; D represents "done". The behaviours of the hammer and mallet are:

$$
\begin{aligned}
&\mathrm{H} \Leftarrow \overline{\mathrm{GH}}\ .\ \overline{\mathrm{PH}}\ .\ \mathrm{H}\\
&\mathrm{M} \Leftarrow \overline{\mathrm{GM}}\ .\ \overline{\mathrm{PM}}\ .\ \mathrm{M}
\end{aligned}
$$

The two men together with the hammer and mallet can be put together to form the workshop:

SHOP = (MAN | MAN | H | M)

The interactions between the men and their tools should not be visible to the outside observer, hence we define

CLOSEDSHOP = SHOP \ L
where L = { GH, PH, GM, PM }.

This agent has sort { IN, $\overline{OUT}$ } and we wish to demonstrate that its behaviour is equivalent to the following specification:

DOINGNOTHING $\Leftarrow$ IN x . ONE(x)
ONE(x) $\Leftarrow$ IN y . TWO(x,y) + $\overline{OUT}$ x . DOINGNOTHING
TWO(x,y) $\Leftarrow$ $\overline{OUT}$ x . ONE(y) + $\overline{OUT}$ y . ONE(x)

(Milner used DOINGONE and DOINGTWO rather than ONE and TWO; those names explain much more clearly what the agent represents but in order to be able to keep each ordered pair of behaviours on a single lines in the manipulation that follows it is necessary to keep the names short. This is also the reason why H and M were used rather than HAMMER and MALLET and D rather than DONE and the labels were given such short names as GH and PM.)

We wish to show that DOINGNOTHING is equivalent to CLOSEDSHOP. It is easy to see that DOINGNOTHING is rigid and totally determinate and it is not difficult to check that CLOSEDSHOP cannot diverge. Hence we apply the algorithm and commence by putting <DOINGNOTHING,CLOSEDSHOP> into R. As each necessary new ordered pair of behaviours is introduced into R it will be indexed with a number and the analysis of ordered pairs by step (iii) of the algorithm will be applied in the order of this index. As it is observed that READY(x) can accept a τ-experiment if x is easy, a GH-experiment unless x is easy and a GM-experiment if x is

neither easy nor hard, these are the properties of jobs that we need to be able to recognise. Hence for convenience of notation we shall define predicates to express these properties:

e(x)	x is easy
ne(x)	x is not easy
n(x)	x is neither easy nor hard

It should be noted that n(x) implies ne(x). Additional notation that is used during the application of the algorithm will be explained as it is introduced. We proceed as follows:

1 <DOINGNOTHING,CLOSEDSHOP>

There are two pairs to put into R:

<ONE(x),(READY(x)|MAN|H|M)\L> (2)

<ONE(x),(MAN|READY(x)|H|M)\L> (3)

(Each pair is actually a class of pairs in which each element is a pair together with an instantiation of a value to the free variable x. It will be assumed that x can take any value unless any constraints are imposed. The (2) and (3) are the indices given to the new pairs that have to be checked)

2 <ONE(x),(READY(x)|MAN|H|M)\L>

Here the pairs that have to be inserted into R depend on the job x, hence classes of pairs are inserted together with predicates as introduced above.

<TWO(x,y),(READY(x)|READY(y)|H|M)\L> (4)

<DOINGNOTHING,CLOSEDSHOP> (=1)

(this comes from the case e(x) but as x does not occur in the resulting behaviours there is no point in writing it alongside the pair. The (=1) represents the fact that this pair is already present in R indexed as 1)

<ONE(x),(PH.D(x)|MAN|$\overline{\text{PH}}$.H|M)\L> [ne(x)] (5)

(here we have written ne(x) alongside the pair to indicate that we need this pair in R only if x is not easy)

$\langle ONE(x), (PM.D(x) | MAN | H | \overline{PM}.M) \backslash L \rangle$ [n(x)] (6)

3 $\langle ONE(x), (MAN | READY(x) | H | M) \backslash L \rangle$

This pair has properties of symmetry relative to the previous pair, hence it is possible to write down the appropriate pairs to be introduced into R without any effort:

$\langle TWO(x,y), (READY(y) | READY(x) | H | M) \backslash L \rangle$ (7)

$\langle DOINGNOTHING, CLOSEDSHOP \rangle$ (=1)

$\langle ONE(x), (MAN | PH.D(x) | \overline{PH}.H | M) \backslash L \rangle$ [ne(x)] (8)

$\langle ONE(x), (MAN | PM.D(x) | H | \overline{PM}.M) \backslash L \rangle$ [n(x)] (9)

4 $\langle TWO(x,y), (READY(x) | READY(y) | H | M) \backslash L \rangle$

From now we shall proceed to write down the appropriate pairs obtained at each stage without any unnecessary explanation.

$\langle ONE(y), (MAN | READY(y) | H | M) \backslash L \rangle$ (=3r)

(This pair is the same as (3) except for the renaming of a free variable; hence the class of pairs under consideration is exactly the same as (3) so there is nothing new to put into R. We use (=3r) to denote this)

$\langle ONE(x), (READY(x) | MAN | H | M) \backslash L \rangle$ (=2)

$\langle TWO(x,y), (PH.D(x) | READY(y) | \overline{PH}.H | M) \backslash L \rangle$ [ne(x)] (10)

$\langle TWO(x,y), (READY(x) | PH.D(y) | \overline{PH}.H | M) \backslash L \rangle$ [ne(y)] (11)

$\langle TWO(x,y), (PM.D(x) | READY(y) | H | \overline{PM}.M) \backslash L \rangle$ [n(x)] (12)

$\langle TWO(x,y), (READY(x) | PM.D(y) | H | \overline{PM}.M) \backslash L \rangle$ [n(y)] (13)

5 $\langle ONE(x), (PH.D(x) | MAN | \overline{PH}.H | M) \backslash L \rangle$ [ne(x)]

$\langle TWO(x,y), (PH.D(x) | READY(y) | \overline{PH}.H | M) \backslash L \rangle$ [ne(x)] (=10)

$\langle ONE(x), (D(x) | MAN | H | M) \backslash L \rangle$ [ne(x)] (14)

6 $\langle ONE(x), (PM.D(x) | MAN | H | \overline{PM}.M) \backslash L \rangle$ [n(x)]

$\langle TWO(x,y), (PM.D(x) | READY(y) | H | \overline{PM}.M) \backslash L \rangle$ [n(x)] (=12)

$\langle ONE(x), (D(x) | MAN | H | M) \backslash L \rangle$ [n(x)] (=14s)

(Here the set of ordered pairs as above such that n(x) is true is a subset of the set such that ne(x) is true (as n(x) implies ne(x)); that set is already present in R (14) so we use the notation (=14s) to indicate that this a subset)

7 $\langle TWO(x,y),(READY(y)|READY(x)|H|M)\backslash L\rangle$

$\langle ONE(y),(READY(y)|MAN|H|M)\backslash L\rangle$ (=2r)

$\langle ONE(x),(MAN|READY(x)|H|M)\backslash L\rangle$ (=3)

$\langle TWO(x,y),(READY(y)|PH.D(x)|\overline{PH}.H|M)\backslash L\rangle$ [ne(x)] (15)

$\langle TWO(x,y),(PH.D(y)|READY(x)|\overline{PH}.H|M)\backslash L\rangle$ [ne(y)] (16)

$\langle TWO(x,y),(READY(y)|PM.D(x)|H|\overline{PM}.M)\backslash L\rangle$ [n(x)] (17)

$\langle TWO(x,y),(PM.D(y)|READY(x)|H|\overline{PM}.M)\backslash L\rangle$ [n(y)] (18)

8 $\langle ONE(x),(MAN|PH.D(x)|\overline{PH}.H|M)\backslash L\rangle$ [ne(x)]

$\langle TWO(x,y),(READY(y)|PH.D(x)|\overline{PH}.H|M)\backslash L\rangle$ [ne(x)] (=15)

$\langle ONE(x),(MAN|D(x)|H|M)\backslash L\rangle$ [ne(x)] (19)

9 $\langle ONE(x),(MAN|PM.D(x)|H|\overline{PM}.M)\backslash L\rangle$ [n(x)]

$\langle TWO(x,y),(READY(y)|PM.D(x)|H|\overline{PM}.M)\backslash L\rangle$ (=17)

$\langle ONE(x),(MAN|D(x)|H|M)\backslash L\rangle$ [n(x)] (=19s)

10 $\langle TWO(x,y),(PH.D(x)|READY(y)|\overline{PH}.H|M)\backslash L\rangle$ [ne(x)]

$\langle ONE(x),(PH.D(x)|MAN|\overline{PH}.H|M)\backslash L\rangle$ [ne(x)] (=5)

$\langle TWO(x,y),(D(x)|READY(y)|H|M)\backslash L\rangle$ [ne(x)] (20)

$\langle TWO(x,y),(PH.D(x)|PM.D(y)|\overline{PH}.H|\overline{PM}.M)\backslash L\rangle$ [ne(x),n(y)] (21)

11 $\langle TWO(x,y),(READY(x)|PH.D(y)|\overline{PH}.H|M)\backslash L\rangle$ [ne(y)]

$\langle ONE(y),(MAN|PH.D(y)|\overline{PH}.H|M)\backslash L\rangle$ [ne(y)] (=8r)

$\langle TWO(x,y),(READY(x)|D(y)|H|M)\backslash L\rangle$ [ne(y)] (22)

$\langle TWO(x,y),(PM.D(x)|PH.D(y)|\overline{PH}.H|\overline{PM}.M)\backslash L\rangle$ [n(x),ne(y)] (23)

12 $\langle TWO(x,y),(PM.D(x)|READY(y)|H|\overline{PM}.M)\backslash L\rangle$ [n(x)]

$\langle ONE(x),(PM.D(x)|MAN|H|\overline{PM}.M)\backslash L\rangle$ [n(x)] (=6)

$\langle TWO(x,y),(PM.D(x)|PH.D(y)|\overline{PH}.H|\overline{PM}.M)\backslash L\rangle$ [n(x),ne(y)] (=23)

$\langle TWO(x,y),(D(x)|READY(y)|H|M)\backslash L\rangle$ [n(x)] (=20s)

13 $\langle TWO(x,y),(READY(x)|PM.D(y)|H|\overline{PM}.M)\backslash L\rangle$ [n(y)]

$\langle ONE(y),(MAN|PM.D(y)|H|\overline{PM}.M)\backslash L\rangle$ [n(y)] (=9r)

$\langle TWO(x,y),(PH.D(x)|PM.D(y)|\overline{PH}.H|\overline{PM}.M)\backslash L\rangle$ [ne(x),n(y)] (=21)

$\langle TWO(x,y),(READY(x)|D(y)|H|M)\backslash L\rangle$ [n(y)] (=22s)

14 <ONE(x),(D(x)|MAN|H|M)\L> [ne(x)]
<TWO(x,y),(D(x)|READY(y)|H|M)\L> [ne(x)] (=20)
<DOINGNOTHING,CLOSEDSHOP> (=10)

15 <TWO(x,y),(READY(y)|PH.D(x)|$\overline{PH}$.H|M)\L> [ne(x)]
<ONE(x),(MAN|PH.D(x)|$\overline{PH}$.H|M)\L> [ne(x)] (=8)
<TWO(x,y),(READY(y)|D(x)|H|M)\L> [ne(x)] (24)
<TWO(x,y),(PM.D(y)|PH.D(x)|$\overline{PH}$.H|$\overline{PM}$.M)\L> [ne(x),n(y)] (25)

16 <TWO(x,y),(PH.D(y)|READY(x)|$\overline{PH}$.H|M)\L> [ne(y)]
<ONE(x),(PH.D(y)|MAN|$\overline{PH}$.H|M)\L> [ne(y)] (=5r)
<TWO(x,y),(D(y)|READY(x)|H|M)\L> [ne(y)] (26)
<TWO(x,y),(PH.D(y)|PM.D(x)|$\overline{PH}$.H|$\overline{PM}$.M)\L> [n(x),ne(y)] (27)

17 <TWO(x,y),(READY(y)|PM.D(x)|H|$\overline{PM}$.M)\L> [n(x)]
<ONE(x),(MAN|PM.D(x)|H|$\overline{PM}$.M)\L> [n(x)] (=9)
<TWO(x,y),(PH.D(y)|PM.D(x)|$\overline{PH}$.H|$\overline{PM}$.M)\L> [n(x),ne(y)] (=27)
<TWO(x,y),(READY(y)|D(x)|H|M)\L> [n(x)] (=24s)

18 <TWO(x,y),(PM.D(y)|READY(x)|H|$\overline{PM}$.M)\L> [n(y)]
<ONE(y),(PM.D(y)|MAN|H|$\overline{PM}$.M)\L> [n(y)] (=6r)
<TWO(x,y),(PM.D(y)|PH.D(x)|$\overline{PH}$.H|$\overline{PM}$.M)\L> [ne(x),n(y)] (=25)
<TWO(x,y),(D(y)|READY(x)|H|M)\L> [n(y)] (=26s)

19 <ONE(x),(MAN|D(x)|H|M)\L> [ne(x)]
<TWO(x,y),(READY(y)|D(x)|H|M)\L> [ne(x)] (=24)
<DOINGNOTHING,CLOSEDSHOP> (=1)

20 <TWO(x,y),(D(x)|READY(y)|H|M)\L> [ne(x)]
<ONE(y),(MAN|READY(y)|H|M)\L> (=3r)
<ONE(x),(D(x)|MAN|H|M)\L> [ne(x)] (=14)
<TWO(x,y),(D(x)|PH.D(y)|$\overline{PH}$.H|M)\L> [ne(x),ne(y)] (28)
<TWO(x,y),(D(x)|PM.D(y)|H|$\overline{PM}$.M)\L> [ne(x),n(y)] (29)

21 <TWO(x,y),(PH.D(x)|PM.D(y)|$\overline{PH}$.H|$\overline{PM}$.M)\L> [ne(x),n(y)]
<TWO(x,y),(D(x)|PM.D(y)|H|$\overline{PM}$.M)\L> [ne(x),n(y)] (=29)
<TWO(x,y),(PH.D(x)|D(y)|$\overline{PH}$.H|M)\L> [ne(x),n(y)] (30)

22 <TWO(x,y),(READY(x)|D(y)|H|M)\L> [ne(y)]
<ONE(y),(MAN|D(y)|H|M)\L> [ne(y)] (=19r)
<ONE(x),(READY(x)|MAN|H|M)\L> (=2)
<TWO(x,y),(PH.D(x)|D(y)|$\overline{PH}$.H|M)\L> [ne(x),ne(y)] (31)
(Here it is observed that the class of pairs (30), which was already inserted into R and "marked for checking" is a subclass of (31); hence it will be unnecessary to check pair (30) separately and we can "unmark" it)
<TWO(x,y),(PM.D(x)|D(y)|H|$\overline{PM}$.M)\L> [n(x),ne(y)] (32)

23 <TWO(x,y),(PM.D(x)|PH.D(y)|$\overline{PH}$.H|$\overline{PM}$.M)\L> [n(x),ne(y)]
<TWO(x,y),(PM.D(x)|D(y)|H|$\overline{PM}$.M)\L> [n(x),ne(y)] (=32)
<TWO(x,y),(D(x)|PH.D(y)|$\overline{PH}$.H|M)\L> [n(x),ne(y)] (=28s)

24 <TWO(x,y),(READY(y)|D(x)|H|M)\L> [ne(x)]
<ONE(x),(MAN|D(x)|H|M)\L> [ne(x)] (=19)
<ONE(y),(READY(y)|MAN|H|M)\L> (=2r)
<TWO(x,y),(PH.D(y)|D(x)|$\overline{PH}$.H|M)\L> [ne(x),ne(y)] (33)
<TWO(x,y),(PM.D(y)|D(x)|H|$\overline{PM}$.M)\L> [ne(x),n(y)] (34)

25 <TWO(x,y),(PM.D(y)|PH.D(x)|$\overline{PH}$.H|$\overline{PM}$.M)\L> [ne(x),n(y)]
<TWO(x,y),(PM.D(y)|D(x)|H|$\overline{PM}$.M)\L> [ne(x),n(y)] (=34)
<TWO(x,y),(D(y)|PH.D(x)|$\overline{PH}$.H|M)\L> [ne(x),n(y)] (35)

26 <TWO(x,y),(D(y)|READY(x)|H|M)\L> [ne(y)]
<ONE(x),(MAN|READY(x)|H|M)\L> (=3)
<ONE(y),(D(y)|MAN|H|M)\L> [ne(y)] (=14r)
<TWO(x,y),(D(y)|PH.D(x)|$\overline{PH}$.H|M)\L> [ne(x),ne(y)] (36)
(Here it is observed that (35) is a subclass of (36); hence there will be no need to check (35))
<TWO(x,y),(D(y)|PM.D(x)|H|$\overline{PM}$.M)\L> [n(x),ne(y)] (37)

27 <TWO(x,y),(PH.D(y)|PM.D(x)|$\overline{PH}$.H|$\overline{PM}$.M)\L> [n(x),ne(y)]
<TWO(x,y),(D(y)|PM.D(x)|H|$\overline{PM}$.M)\L> [n(x),ne(y)] (=37)
<TWO(x,y),(PH.D(y)|D(x)|$\overline{PH}$.H|M)\L> [n(x),ne(y)] (=33s)

28 <TWO(x,y),(D(x)|PH.D(y)|$\overline{PH}$.H|M)\L> [ne(x),ne(y)]
<ONE(y),(MAN|PH.D(y)|$\overline{PH}$.H|M)\L> [ne(y)] (=8r)
<TWO(x,y),(D(x)|D(y)|H|M)\L> [ne(x),ne(y)] (38)

29 <TWO(x,y),(D(x)|PM.D(y)|H|$\overline{PM}$.M)\L> [ne(x),n(y)]
<ONE(y),(MAN|PM.D(y)|H|$\overline{PM}$.M)\L> [n(y)] (=9r)
<TWO(x,y),(D(x)|D(y)|H|M)\L> [ne(x),n(y)] (=38s)

31 <TWO(x,y),(PH.D(x)|D(y)|$\overline{PH}$.H|M)\L> [ne(x),ne(y)]
<ONE(x),(PH.D(x)|MAN|$\overline{PH}$.H|M)\L> [ne(x)] (=5)
<TWO(x,y),(D(x)|D(y)|H|M)\L> [ne(x),ne(y)] (=38)

32 <TWO(x,y),(PM.D(x)|D(y)|H|$\overline{PM}$.M)\L> [n(x),ne(y)]
<ONE(x),(PM.D(x)|MAN|H|$\overline{PM}$.M)\L> [n(x)] (=6)
<TWO(x,y),(D(x)|D(y)|H|M)\L> [n(x),ne(y)] (=38s)

33 <TWO(x,y),(PH.D(y)|D(x)|$\overline{PH}$.H|M)\L> [ne(x),ne(y)]
<ONE(y),(PH.D(y)|MAN|$\overline{PH}$.H|M)\L> [ne(y)] (=5r)
<TWO(x,y),(D(y)|D(x)|H|M)\L> [ne(x),ne(y)] (39)

34 <TWO(x,y),(PM.D(y)|D(x)|H|$\overline{PM}$.M)\L> [ne(x),n(y)]
<ONE(y),(PM.D(y)|MAN|H|$\overline{PM}$.M)\L> [n(y)] (=6r)
<TWO(x,y),(D(y)|D(x)|H|M)\L> [ne(x),n(y)] (=39s)

36 <TWO(x,y),(D(y)|PH.D(x)|$\overline{PH}$.H|M)\L> [ne(x),ne(y)]
<ONE(x),(MAN|PH.D(x)|$\overline{PH}$.H|M)\L> [ne(x)] (=8)
<TWO(x,y),(D(y)|D(x)|H|M)\L> [ne(x),ne(y)] (39)

37 <TWO(x,y),(D(y)|PM.D(x)|H|$\overline{PM}$.M)\L> [n(x),ne(y)]
<ONE(x),(MAN|PM.D(x)|H|$\overline{PM}$.M)\L> [n(x)] (=9)
<TWO(x,y),(D(y)|D(x)|H|M)\L> [n(x),ne(y)] (=39s)

38 <TWO(x,y),(D(x)|D(y)|H|M)\L> [ne(x),ne(y)]
<ONE(y),(MAN|D(y)|H|M)\L> [ne(y)] (=19r)
<ONE(x),(D(x)|MAN|H|M)\L> [ne(x)] (=14)

39 <TWO(x,y),(D(y)|D(x)|H|M)\L> [ne(x),ne(y)]
<ONE(x),(MAN|D(x)|H|M)\L> [ne(x)] (=19)
<ONE(y),(D(y)|MAN|H|M)\L> [ne(y)] (=14r)

All the pairs that were inserted into R have been shown to satisfy the conditions of proposition 5.6, so it follows that $R \in E(R)$, and hence that DOINGNOTHING $\approx$ CLOSEDSHOP as required.

For this example no alternative method using the axiomatic approach with $\approx$ has been found that will demonstrate the equivalence of the constructed behaviour CLOSEDSHOP and its specification DOINGNOTHING. Hence, although the application of the algorithm required a lot of time and space, no quicker technique is known that will prove this equivalence. This helps to indicate the usefulness of the fixed-point definition of equivalence.

Having studied an example it is not difficult to see how the algorithm for pure CCS needs to be extended for the full calculus. The major difference is that when a pair with free variables is inserted into the set R it represents a class of pairs of behaviours corresponding to the possible instantiations of values to the free variables. Hence the naming of the variables is irrelevant, and when checking whether a pair is already present in the set any variables occurring in the expressions must be treated accordingly. Furthermore it is often necessary to associate with a pair of behaviours a set of conditions on the free variables involved. In implementing the algorithm, in order to avoid unnecessary checking, it should be noted whether the truth of any of the predicates under consideration implies the truth of any others; in the example as n(x) implied ne(x) it was frequently possible to ignore a pair together with the condition ne(x) as the

same pair with the condition n(x) was already in the relation R.

The question of under what conditions the algorithm can be guaranteed to terminate may become more complex in the full calculus. If values are allowed to be drawn from infinite sets even very simple behaviours need not have finite classes of derivatives, for example $\alpha x.\bar{\beta}x.NIL$ has infinitely many different derivatives if the set of values that the variable x may take is not finite. However, the fact that the steps of the algorithm can be applied to classes of behaviours that satisfy the same predicates (such as n, e and ne in the example) may solve some of the problems that could be encountered.

In conclusion, in this chapter it has been demonstrated that an alternative definition of observational equivalence as a maximal fixed-point can be used and that this leads to a stronger equivalence than the original one. To demonstrate the equivalence of two agents under the new definition it has been shown to be adequate to demonstrate the existence of a bisimulation between the agents and it has been demonstrated that in order to do this only single-length derivations need be considered. In some cases bisimulations can be presented almost trivially, even when any other proof of equivalence would be difficult or impossible, demonstrating the power of the technique. An algorithm for constructing a bisimulation under certain reasonable conditions on the behaviours has been given, but further research is required into restrictions to guarantee its termination.

6 OBSERVATION CONFLUENCE AS A MAXIMAL FIXED POINT

INTRODUCTION

In chapter 3 we examined confluence and determinacy for the full CCS. The work in that chapter was restricted to strong confluence and determinacy which were defined relative to $\xrightarrow{\mu}$ derivations and preserved by strong congruence, $\sim$, but not by observational equivalence. Milner in [Mil4] defined and examined briefly a concept of observation confluence and determinacy, using $\overset{\varepsilon}{\Rightarrow}$ derivations, which was preserved by $\approx$. The work there involved the definition of the excess of one derivation string over another, which was (approximately) the string of elements occuring in one string but not in the other. Here we will define observation confluence and determinacy relative to the maximal fixed-point equivalence $\approx_F$ (introduced in the previous chapter) and since it has been shown that defining the equivalence using derivation strings in $\Lambda \cup \{\varepsilon\}$ gives the same result as using strings in Λ^* we need work only with derivation strings of length 0 or 1 thus avoiding this concept of excess. As $\approx_F$ was defined as a fixed point rather than the limit of a decreasing sequence of equivalences it is logical when defining confluence relative to $\approx_F$ to use the maximal fixed point of a defining relation rather than the limit of a decreasing sequence of sets of k-confluent behaviours. Having defined and examined observation confluence and determinacy for the pure CCS we shall show how it can be extended to the full calculus and give an example of its use in the proof of observational equivalence for a behaviour not in the derived calculus CCCS introduced in chapter 3.

OBSERVATION CONFLUENCE AND DETERMINACY FOR PURE CCS

In order to define observation confluence and determinacy as a maximal fixed point it is first necessary to present an

appropriate defining relation. This is simply an adaptation of Milner's definition from chapter 10 of [Mil4] using derivations strings of unit length only. We shall call this relation ϕ.

Definition

Given a set R of behaviours in the pure CCS, $\phi(R)$ is defined to be the set of behaviours A such that:

(i) $A \stackrel{s}{\Rightarrow} B$ implies $B \in R$ for $s \in \Lambda \cup \{\epsilon\}$;

(ii) $\begin{matrix} & \stackrel{s}{\Rightarrow} B \\ A & \\ & \stackrel{s}{\Rightarrow} C \end{matrix}$ implies $\begin{matrix} B \stackrel{t}{\Rightarrow} D \\ \\ C \stackrel{t}{\Rightarrow} E \end{matrix}$ $\begin{matrix} \\ R \\ F \end{matrix}$ for $s \in \Lambda \cup \{\epsilon\}$; and

(iii) $\begin{matrix} & \stackrel{s}{\Rightarrow} B \\ A & \\ & \stackrel{s'}{\Rightarrow} C \end{matrix}$ implies $\begin{matrix} B \stackrel{t'}{\Rightarrow} D \\ \\ C \stackrel{t}{\Rightarrow} E \end{matrix}$ $\begin{matrix} \\ \mathcal{R} \\ F \end{matrix}$ for $s, s' \in \Lambda \cup \{\epsilon\}$ such that $s \neq s'$.

This definition is analogous to the one used by Milner to define OCD_{k+1} from OCD_k; the second clause implies determinacy, and the second and third clauses together give confluence. It would be possible to define our observation confluence and determinacy using a sequence of sets of k-confluent behaviours; however as we are working with a fixed-point definition of observational equivalence it is logical to define the set of (observation-) confluent and determinate agents to be the maximal fixed point of the relation ϕ under the ordering of set inclusion:

Definition

CONFL, the set of observation-confluent and -determinate agents, is defined by

$$\text{CONFL} = \cup\{ S : S \subseteq \phi(S) \}.$$

It is easily seen that ϕ is monotonic, so by proposition 5.1 we can deduce that the above definition does indeed give the maximal fixed point of ϕ. Using this definition, in order to prove that a behaviour A is observation-confluent it is sufficient to demonstrate the existence of a set S such that $A \in S$ and $S \subseteq \phi(S)$. We shall use this technique to show that observation confluence is preserved by $\approx_F$.

Proposition 6.1

If $A \in \text{CONFL}$ and $A \approx_F A'$ then $A' \in \text{CONFL}$.

Proof

We have to define a set T such that $A' \in T$ and $T \subseteq \phi(T)$. We choose

$$T = \{ y : x \approx_F y \text{ for some } x \in \text{CONFL} \}$$

then clearly $A' \in T$. To demonstrate that $T \subseteq \phi(T)$ we assume $y \in T$ and have to show that y satisfies the three clauses of the definition of $\phi(T)$. As $y \in T$ there is some $x \in \text{CONFL}$ such that $x \approx_F y$.

(i) We assume that $y \stackrel{s}{\Rightarrow} z$ for $s \in A^* \cup \{\epsilon\}$ and have to show that $z \in T$. As $x \approx_F y$ and $y \stackrel{s}{\Rightarrow} z$ we can deduce that $x \stackrel{s}{\Rightarrow} w$ for some w such that $w \approx_F z$. Then as $x \in \text{CONFL} = \phi(\text{CONFL})$ we have $w \in \text{CONFL}$, hence $z \in T$ as required.

(ii) We assume that $y \stackrel{s}{\Rightarrow} y_1$ and $y \stackrel{s}{\Rightarrow} y_2$ for $s \in A^* \cup \{\epsilon\}$ and have to show that $y_1 \stackrel{\epsilon}{\Rightarrow} z_1$ and $y_2 \stackrel{\epsilon}{\Rightarrow} z_2$ for some z_1 and z_2

such that $z_1 \simeq_F z_2$. As $x \simeq_F y$ there exist x_1 and x_2 such that $x \stackrel{s}{\Rightarrow} x_1$ with $x_1 \simeq_F y_1$ and $x \stackrel{s}{\Rightarrow} x_2$ with $x_2 \simeq_F y_2$. Then as $x \in$ CONFL it follows that $x_1 \stackrel{t}{\Rightarrow} w_1$ and $x_2 \stackrel{t}{\Rightarrow} w_2$ with $w_1 \simeq_F w_2$. Then as $x_1 \simeq_F y_1$ we have $y_1 \stackrel{t}{\Rightarrow} z_1$ for some z_1 such that $w_1 \simeq_F z_1$, and likewise $y_2 \stackrel{t}{\Rightarrow} z_2$ for some z_2 such that $w_2 \simeq_F z_2$. It remains to show that $z_1 \simeq_F z_2$, but this is trivial as $z_1 \simeq_F w_1 \simeq_F w_2 \simeq_F z_2$.

(iii) We assume that $y \stackrel{s_1}{\Rightarrow} y_1$ for $s_1 \in \Lambda \cup \{\varepsilon\}$ and that $y \stackrel{s_2}{\Rightarrow} y_2$ for $s_2 \in \Lambda \cup \{\varepsilon\}$ and furthermore that $s_1 \neq s_2$ and have to show that $y_1 \stackrel{s_2}{\Rightarrow} z_1$ and $y_2 \stackrel{s_1}{\Rightarrow} z_2$ for some z_1 and z_2 such that $z_1 \simeq_F z_2$. This is done in exactly the same way as (ii) above.

It has been shown that $y \in \phi(T)$ for all $y \in T$, hence $T \subseteq \phi(T)$, so $T \subseteq$ CONFL. As $A' \in T$ we conclude that $A' \in$ CONFL as required.

Having proved that observation confluence is preserved by observational equivalence using the maximal fixed-point definition, we have gone some way towards justifying the definition chosen for observation confluence. The next step is to demonstrate that a result analogous to Milner's strong confluence theorem holds for this observation confluence. This will enable the use of the fixed-point confluence in proofs of equivalence in the same way that strong confluence was used earlier.

Theorem 6.2 (Observation confluence theorem for pure CCS)

If $A \in$ CONFL and $A \stackrel{t}{\Rightarrow} B$ then $A \simeq_F B$.

Proof

We define a relation R by

$$R = \{ \langle x,y \rangle : x \in \mathrm{CONFL},\ y \approx_F z \text{ for some } z \text{ such that } x \stackrel{\varepsilon}{\Rightarrow} z \}$$

and demonstrate that this is a bisimulation. This involves proving that for each $\langle x,y \rangle$ in R the two clauses of the definition of E(R) are satisfied:

(i) We assume that $\langle x,y \rangle \in R$ and $x \stackrel{s}{\rightarrow} x'$ for $s \in \Lambda \cup \{\varepsilon\}$ and have to prove that $y \stackrel{s}{\Rightarrow} y'$ for some y' such that $\langle x',y' \rangle \in R$. From the construction of R we know $x \in \mathrm{CONFL}$ and there exists some z such that $x \stackrel{\varepsilon}{\Rightarrow} z$ and $y \approx_F z$. As $x \stackrel{s}{\rightarrow} x'$ and $x \stackrel{\varepsilon}{\Rightarrow} z$ we can deduce that $x' \stackrel{\varepsilon}{\Rightarrow} w$ and $z \stackrel{s}{\Rightarrow} z'$ with $w \approx_F z'$. Then as $y \approx_F z$ and $z \stackrel{s}{\Rightarrow} z'$, it follows that $y \stackrel{s}{\Rightarrow} y'$ for some y' such that $y' \approx_F z'$ and hence such that $y' \approx_F w$. As $x \in \mathrm{CONFL}$ and $x \stackrel{s}{\Rightarrow} x'$ we also know that $x' \in \mathrm{CONFL}$. We have shown that $x' \in \mathrm{CONFL}$, $x' \stackrel{\varepsilon}{\Rightarrow} w$ and $y' \approx_F w$, hence $\langle x',y' \rangle \in R$ as required.

(ii) We assume that $\langle x,y \rangle \in R$ and $y \stackrel{s}{\Rightarrow} y'$ for $s \in \Lambda \cup \{\varepsilon\}$ and have to prove that $x \stackrel{s}{\Rightarrow} x'$ for some x' such that $\langle x',y' \rangle \in R$. From the construction of R we know that $x \in \mathrm{CONFL}$ and $x \stackrel{\varepsilon}{\Rightarrow} z$ for some z such that $y \approx_F z$. As $y \stackrel{s}{\Rightarrow} y'$ we have $z \stackrel{s}{\Rightarrow} x'$ with $y' \approx_F x'$. Then, as $x \stackrel{\varepsilon}{\Rightarrow} z \stackrel{s}{\Rightarrow} x'$ we have $x \stackrel{s}{\Rightarrow} x'$, and as $x \in \mathrm{CONFL}$ we deduce that $x' \in \mathrm{CONFL}$. As we also have $x' \stackrel{\varepsilon}{\Rightarrow} x'$ (trivially) and $y' \approx_F x'$ it follows that $\langle x',y' \rangle \in R$ as required.

This completes the proof that $R \subseteq E(R)$; it is easy to see that $\langle A,B \rangle \in R$, so it follows that $A \approx_F B$ as required.

Having defined observation confluence and determinacy (as a single concept) using the fixed-point approach for pure CCS and proved that it is preserved by observational equivalence and it admits a result analogous to the strong confluence theorem we are ready to extend the definition to the full calculus.

OBSERVATION CONFLUENCE IN THE FULL CALCULUS

In chapter 3 we extended Milner's strong confluence and determinacy to the full CCS with value-passing; we use a similar approach to extend the fixed-point observation confluence which has just been introduced. Recalling chapter 3, we begin with an informal discussion of how the definition of the relation ϕ must be adapted, then we present a formal definition of the set of observation-confluent programs in the full calculus (which can be extended in the obvious way to define observation confluence for agents with free variables). After doing this and observing that proposition 6.1 and theorem 6.2 hold in this wider context we proceed to show with an example how the observation confluence theorem can be used in the proof of observational equivalence of behaviours

We consider first the extension of determinacy to the full calculus in chapter 3. This entailed considering separately derivations of the forms $\stackrel{\alpha v}{\rightarrow}$ and $\stackrel{\bar{\alpha} v}{\rightarrow}$. In view of this we need to split the clause (ii) of the definition of ϕ into two halves in order to extend observation determinacy. We will require clauses of the form

(a) $A \stackrel{\alpha}{\rightarrow} B$, $A \stackrel{\alpha}{\rightarrow} C$ implies $B \stackrel{\varepsilon}{\Rightarrow} D \asymp_F [m/x]U$ and $C \stackrel{\varepsilon}{\Rightarrow} E \asymp_F [n/x]U$ for some U and

(b) $A \stackrel{\alpha}{\rightarrow} B$, $A \stackrel{\alpha}{\rightarrow} C$ implies $m = n$ and $B \stackrel{\varepsilon}{\Rightarrow} D$, $C \stackrel{\varepsilon}{\Rightarrow} E$ $\mathcal{U}_F$.

We should also retain the case $s = \varepsilon$ from the second clause of the definition of ϕ, which will give another clause

(c) $A \overset{\tau}{\Rightarrow} B$, $A \overset{\tau}{\Rightarrow} C$ implies $B \overset{\tau}{\Rightarrow} D$, $C \overset{\tau}{\Rightarrow} E$, $D \approx_F E$.

We now consider the extension of the confluence properties, recalling from chapter 3 that for strong confluence $A \overset{\mu}{\rightarrow} B$ and $A \overset{\nu}{\rightarrow} C$ implies that one of three cases should hold:

either $B \overset{\nu}{\rightarrow} D$ and $C \overset{\mu}{\rightarrow} E$ with $D \sim E$,

or $\mu = \nu$ and $B \sim C$,

or $\mu = \alpha\, m$ and $\nu = \alpha\, n$ with $B \sim [m/x]U$ and $C \sim [n/x]U$ for some behaviour U.

Using this to adapt clause (iii) of the definition of ϕ would give a clause of the form

(d) $A \overset{s_1}{\Rightarrow} B$, $A \overset{s_2}{\Rightarrow} C$ implies either $B \overset{s_1}{\Rightarrow} D$, $C \overset{s_2}{\Rightarrow} E$, $D \approx_F E$ or....

However it should be noted that clauses (a) - (c) above already imply that $B \approx_F C$ if $s_1 = s_2$ and that $B \overset{\tau}{\Rightarrow} D \approx_F [m/x]U$ and $C \overset{\tau}{\Rightarrow} E \approx_F [n/x]U$ if $s_1 = \alpha\, m$ and $s_2 = \alpha\, n$, so all we actually need here is a clause which states that $B \overset{s_1}{\Rightarrow} D$ and $C \overset{s_2}{\Rightarrow} E$ with $D \approx_F E$ when s_1 and s_2 do not satisfy either of the two conditions referred to.

We are now ready to define a relation $\bar{\phi}$ by considering the above adaptations to ϕ. It should be noted that the sets involved will contain only programs and not behaviours with free variables.

Definition

Given a set R of programs, $\bar{\phi}(R)$ is defined to be the set of programs A such that:

(i) $A \xrightarrow{s} B$ implies $B \in R$ for $s \in \Lambda \times \forall \cup \{\epsilon\}$;

(ii) $\begin{matrix} & & B \\ A & \swarrow\!\!\searrow & \\ & & C \end{matrix}$ implies $\begin{matrix} B \xrightarrow{t} D \approx_F [m/x]U \\ C \xrightarrow{t} E \approx_F [n/x]U \end{matrix}$;

(iii) $\begin{matrix} & & B \\ A & \swarrow\!\!\searrow & \\ & & C \end{matrix}$ implies $m = n$ and $\begin{matrix} B \xrightarrow{t} D \\ \approx_F ; \\ C \xrightarrow{t} E \end{matrix}$

(iv) $\begin{matrix} & & B \\ A & \swarrow\!\!\searrow & \\ & & C \end{matrix}$ implies $\begin{matrix} B \xrightarrow{t} D \\ \approx_F ; \text{ and} \\ C \xrightarrow{t} E \end{matrix}$

(v) $\begin{matrix} & & B \\ A & \swarrow\!\!\searrow & \\ & & C \end{matrix}$ implies $\begin{matrix} B \xrightarrow{s'} D \\ \approx_F \text{ for } s,s' \in \Lambda \times \forall \cup \{\epsilon\} \text{ such that none} \\ C \xrightarrow{s} E \end{matrix}$

of the cases (ii) - (iv) above applies.

Having defined the relation $\bar{\Phi}$ on sets of programs we define the set of observation-confluent programs in the full calculus to be its maximal fixed-point.

Definition

CONF, the set of observation-confluent and -determinate programs in full CCS, is defined by

$$CONF = \cup \{ S : S \subseteq \bar{\Phi}(S) \}.$$

This definition does indeed give a maximal fixed point as $\bar{\Phi}$ is easily checked to be monotonic. Having defined $\bar{\Phi}$ and CONF it is

a routine matter to extend propositions 6.1 and 6.2 to the full calculus.

Proposition 6.3

If $A \in CONF$ and $A \approx_F A'$ then $A' \in CONF$.

Theorem 6.4 (Observation confluence theorem)

If $A \in CONF$ and $A \overset{\tau}{\Rightarrow} B$ then $A \approx_F B$.

Proofs

Almost identical to proofs of 6.1 and 6.2.

So far, we have defined observation confluence only on programs (with no free variables) in the full calculus. The extension to behaviours is made in the obvious way:

Definition

A behaviour with free variables in the full CCS is said to be observation-confluent if and only if for all instantiations of values to the free variables the programs obtained are elements of the set CONF.

AN EXAMPLE - A CONFLUENT QUEUE CONSTRUCTION

We are now ready to consider an example of the use of theorem 6.4; it will be seen that the behaviours under consideration are not in the derived calculus CCCS introduced in chapter 3; hence their confluence has to be shown directly; this is where the

definition of CONF as a fixed point proves useful. The example we shall consider is a queue construction; an illustration of its typical behaviour is given on the next page. (As in the some of the diagrams in [Mil4] arrowheads are used to indicate which communications an agent is capable of offering when the queue is in a given state.) It can be seen that input of values occurs at α labels and output at $\bar{\beta}$ labels. To link the agents together we need to define a chaining combinator to perform relabelling and restriction. We define

$$B \frown C = (B[\gamma/\delta] \mid C[\gamma/\beta]) \setminus \gamma$$

and then define the agents FRONT, CELL(x), HEAD(x) and BACK as follows:

$$\begin{aligned}
\text{FRONT} &= \delta.\text{NIL} \\
\text{CELL}(x) &= \beta.\text{HEAD}(x) \\
\text{HEAD}(x) &= \bar{\beta}x.\text{FRONT} \\
\text{BACK} &= \alpha x.(\text{CELL}(x) \frown \text{BACK})
\end{aligned}$$

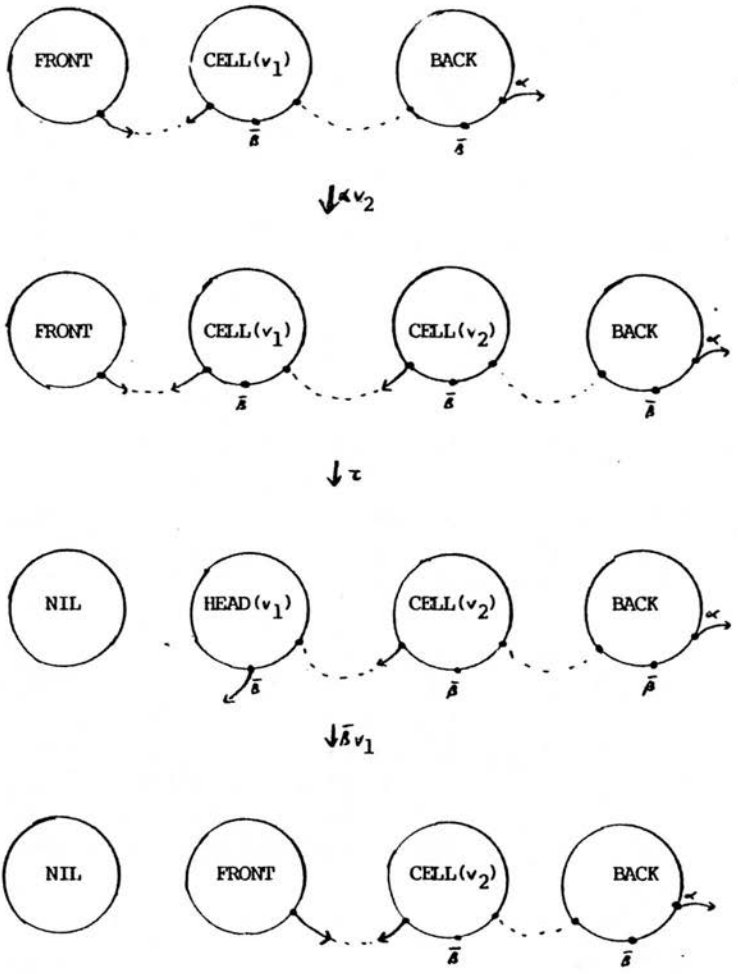
The empty queue is defined:

$$\text{EMPTYQUEUE} = \text{FRONT} \frown \text{BACK}$$

and we wish to prove that EMPTYQUEUE behaves as we would expect. It will be assumed that the values that can be stored in the queue (and hence the values that can be passed by α- and β-communications) are the elements of some set X. Hence in the remainder of this chapter it will be assumed that x and y range over X and s over X^*; as usual ϵ will be used to represent the empty string in X^*. With these assumptions we introduce some extra definitions:

$$\begin{aligned}
\text{CELLS}(\epsilon) &= \text{BACK} \\
\text{CELLS}(ys) &= \text{CELL}(y) \frown \text{CELLS}(s) \\
\text{QUEUE}(s) &= \text{FRONT} \frown \text{CELLS}(s)
\end{aligned}$$

Typical behaviour of the queue construction



It follows that EMPTYQUEUE = QUEUE(ε) and in order to prove that EMPTYQUEUE behaves as one would expect it will be sufficient to show that

$$\text{QUEUE}(\varepsilon) \simeq \alpha x.\text{QUEUE}(x) \text{ and}$$
$$\text{QUEUE}(ys) \simeq \alpha x.\text{QUEUE}(ysx) + \bar{\beta} y.\text{QUEUE}(s).$$

At this stage it should be noted that it is easy to check that CELLS(s) has sort $\{\alpha, \bar{\beta}, \gamma\}$ and QUEUE(s) has sort $\{\alpha, \bar{\beta}\}$.

We commence the proof that QUEUE satisfies these equivalences by establishing an intermediate result that

$$\text{CELLS}(ys) = \gamma.(\text{HEAD}(y) \frown \text{CELLS}(s)) + \alpha x.\text{CELLS}(ysx)$$

by using induction on the length of the string s. For the base step of the induction s = ε and we have

$$\begin{aligned}
\text{CELLS}(y) &= \text{CELL}(y) \frown \text{BACK} \\
&= (\gamma.\text{HEAD}(y)) \frown \alpha x.(\text{CELL}(x) \frown \text{BACK}) \\
&= \gamma.(\text{HEAD}(y) \frown \alpha x.(\text{CELL}(x) \frown \text{BACK})) \\
&\quad + \alpha x.((\gamma.\text{HEAD}(y) \frown (\text{CELL}(x) \frown \text{BACK})) \\
&= \gamma.(\text{HEAD}(y) \frown \text{BACK}) + \alpha x.(\text{CELL}(y) \frown \text{CELL}(x)) \\
&= \gamma.(\text{HEAD}(y) \frown \text{CELLS}(\varepsilon)) + \alpha x.\text{CELLS}(yx).
\end{aligned}$$

For the inductive step we assume the result for s and prove it for zs ($z \in X$):

$$\begin{aligned}
\text{CELLS}(yzs) &= \text{CELL}(y) \frown \text{CELLS}(zs) \\
&= (\gamma.\text{HEAD}(y)) \frown (\gamma.(\text{HEAD}(z) \frown \text{CELLS}(s)) + \alpha x.\text{CELLS}(zsx)) \\
&= \gamma.(\text{HEAD}(y) \frown (\gamma.(\text{HEAD}(z) \frown \text{CELLS}(s)) + \alpha x.\text{CELLS}(zsx))) \\
&\quad + \alpha x.((\gamma.\text{HEAD}(y) \frown \text{CELLS}(zsx)) \\
&= \gamma.(\text{HEAD}(y) \frown \text{CELLS}(zs)) + \alpha x.(\text{CELL}(y) \frown \text{CELLS}(zsx)) \\
&= \gamma.(\text{HEAD}(y) \frown \text{CELLS}(zs)) + \alpha x.\text{CELLS}(yzsx)
\end{aligned}$$

Having established the intermediate result we wish to prove that QUEUE satisfies the two equivalences stated earlier. The use of the chaining operator $\frown$ involves composition of agents whose sorts are non-disjoint; hence this operator is not in the derived calculus CCCS introduced in chapter 3. Hence we choose to apply confluence by using the observation confluence theorem, so it is necessary to show first that QUEUE(s) $\in$ CONF for $s \in X^*$. To do this we need to find a set S such that QUEUE(s) $\in$ S and $S \leq \bar{\Phi}(S)$. We start by defining a set T:

$$T = \{ \text{QUEUE}(s), \text{HEAD}(y) \frown \text{CELLS}(s) : s \in X^*, y \in X \}$$

It is easy to check (using the intermediate result to see how CELLS(s) behaves) that the last four clauses of the definition of $\bar{\Phi}(T)$ are satisfied by all elements of T; however it is not the case that $T \leq \bar{\Phi}(T)$ as the first clause is not satisfied; this is because

$$\text{FRONT} \frown \text{CELLS}(xs) \xrightarrow{\varepsilon} \text{NIL} \frown (\text{HEAD}(x) \frown \text{CELLS}(s)).$$

However it is clear that $\text{NIL} \frown A \approx_F A$ for all $A \in T$; hence we define S to be the set comprising all elements of T with an arbitrary number of NILs chained onto the front, then $S \leq \bar{\Phi}(S)$, hence QUEUE(s) $\in$ CONF. We now proceed to show that QUEUE satisfies the two equivalences expressed earlier and can apply theorem 6.4 where appropriate.

The first equivalence to prove is that $\text{QUEUE}(\varepsilon) \simeq \bar{\delta}\, x.\text{QUEUE}(x)$; this does not actually require the use of the confluence theorem and is quite routine:

$$\begin{aligned}
\text{QUEUE}(\varepsilon) &= \text{FRONT} \frown \text{CELLS}(\varepsilon) \\
&= \text{FRONT} \frown \text{BACK} \\
&= (\varepsilon.\text{NIL}) \frown (\ll x.(\text{CELL}(x) \frown \text{BACK})) \\
&= \ll x.((\varepsilon.\text{NIL}) \frown (\text{CELL}(x) \frown \text{BACK}))
\end{aligned}$$

$$= \alpha x.(\mathrm{FRONT} \frown \mathrm{CELLS}(x))$$
$$= \alpha x.\mathrm{QUEUE}(x)$$

The proof of the second equivalence is much longer; we have to show that

$$\mathrm{QUEUE}(ys) \approx \alpha x.\mathrm{QUEUE}(ysx) + \bar{\delta} y.\mathrm{QUEUE}(s).$$

From the definition $\mathrm{QUEUE}(ys) = \mathrm{FRONT} \frown \mathrm{CELLS}(ys)$ and we deduce from the intermediate result about CELLS that

$$\mathrm{CELLS}(ys) \overset{\varepsilon}{\Rightarrow} \mathrm{HEAD}(y) \frown \mathrm{CELLS}(s).$$

Additionally from the definition, $\mathrm{FRONT} \overset{\delta}{\Rightarrow} \mathrm{NIL}$, hence

$$\mathrm{QUEUE}(ys) \overset{\varepsilon}{\Rightarrow} \mathrm{NIL} \frown (\mathrm{HEAD}(y) \frown \mathrm{CELLS}(s)).$$

As QUEUE(ys) is in CONF we can apply the observation confluence theorem which implies that

$$\mathrm{QUEUE}(ys) \approx \mathrm{NIL} \frown (\mathrm{HEAD}(y) \frown \mathrm{CELLS}(s))$$
$$= \mathrm{HEAD}(y) \frown \mathrm{CELLS}(s).$$

Here we have two separate cases, depending on whether $s = \varepsilon$. In the case $s = \varepsilon$ we proceed

$$\begin{aligned}
\mathrm{QUEUE}(y) &\approx \mathrm{HEAD}(y) \frown \mathrm{BACK} \\
&= (\bar{\delta} y.\mathrm{FRONT}) \frown \alpha x.(\mathrm{CELL}(x) \frown \mathrm{BACK}) \\
&= \bar{\delta} y.(\mathrm{FRONT} \frown \alpha x.(\mathrm{CELL}(x) \frown \mathrm{BACK})) \\
&+ \alpha x.((\bar{\delta} y.\mathrm{FRONT}) \frown (\mathrm{CELL}(x) \frown \mathrm{BACK})) \\
&= \bar{\delta} y.(\mathrm{FRONT} \frown \mathrm{BACK}) + \alpha x.(\mathrm{HEAD}(y) \frown \mathrm{CELLS}(x)), \\
&= \bar{\delta} y.\mathrm{QUEUE}(\varepsilon) + \alpha x.\mathrm{QUEUE}(yx) \\
&\qquad \text{(using } \mathrm{QUEUE}(ys) \approx \mathrm{HEAD}(y) \frown \mathrm{CELLS}(x)) \\
&= \bar{\delta} y.\mathrm{QUEUE}(s) + \alpha x.\mathrm{QUEUE}(ysx) \quad (\text{as } s = \varepsilon)
\end{aligned}$$

and this is the required result. In the case $|s| > 1$, we may write

s=zs' and proceed (using the intermediate result for CELLS)

$$
\begin{aligned}
QUEUE(ys) &\simeq HEAD(y) \frown CELLS(zs') \\
&= (\tilde{\delta}y.FRONT) \frown (\tilde{\nu}.(HEAD(z) \frown CELLS(s')) + \alpha x.CELLS(zs'x)) \\
&= \tilde{\beta}y.(FRONT \frown (\tilde{\nu}.(HEAD(z) \frown CELLS(s')) + \alpha x.CELLS(zs'x))) \\
&\quad + \alpha x.((\tilde{\delta}y.FRONT) \frown CELLS(zs'x)) \\
&= \tilde{\beta}y.(FRONT \frown CELLS(zs') + \alpha x.(HEAD(y) \frown CELLS(zs'x)) \\
&= \tilde{\beta}y.QUEUE(zs') + \alpha x.QUEUE(yzs'x) \\
&\qquad \text{(using } QUEUE(ys) \simeq HEAD(y) \frown CELLS(x)) \\
&= \tilde{\beta}y.QUEUE(s) + \alpha x.QUEUE(ysx)
\end{aligned}
$$

as required. This completes the proof that QUEUE behaves as one should expect; the property proved using the observation confluence theorem was used three times (once when derived and twice subsequently). Without this result much lengthier expressions would have been involved as the use of the expansion theorem would have been greater.

It is felt that this queue construction should be strongly confluent under the definition given in chapter 3; however since it is not in CCCS, to prove it strongly confluent would require proving k-confluence inductively for $k \geq 0$; the definition of observation confluence we have used makes such induction unnecessary.

In conclusion, it has been seen that the use of the maximal fixed-point definition of observational equivalence simplifies considerably the presentation of observation confluence, and therefore the extension from pure CCS to the full calculus can be expressed without undue difficulty. The ability to demonstrate the confluence of behaviours by constructing sets $S \subseteq \tilde{\mathcal{E}}(S)$ permits the application of the observation confluence theorem to a much wider class of behaviours than the derived calculus CCCS introduced earlier, to which the strong confluence theorem could be applied. Therefore the observation confluence theorem is seen to be a more useful result for general application, but the work

on strong confluence has not been devalued, since the strong confluence theorem can be applied to agents in CCCS directly without having to first confirm that the behaviours under consideration are in fact confluent.

7 CONCLUDING REMARKS

A number of different results have been proved in this thesis and applied to proofs of observational equivalence of behaviour expressions in CCS. It is hoped that these results, along with future extensions that have been suggested, will lead to a useful repertoire of techniques for producing reasonably short equivalence proofs within the calculus, and that this might enable the study of larger systems of agents than has been so far attempted. The examples considered have demonstrated that the results presented can be of considerable use; it is anticipated that in considering larger examples some combinations of the techniques introduced may be applied.

There are several limitations on the application of the results presented in the preceding chapters; one of the major ones appears to be that the criteria for uniqueness of solution of recursive equations have been studied only for pure CCS, and no attempt has yet been made to consider transformations other than simple contexts F[ ] for which the behaviour identifier b occurs only once in the expression denoting F[b]. It is clear that further work is needed in this field if the property of uniqueness is to be used in proofs about agents modelling real systems, where behaviour expressions are necessarily more complex than the limited class considered. Some indications have been given as to how further development of the theory may be approached; it would probably be appropriate to attempt to adapt the results already presented to the fixed-point equivalence $\approx_F$. It is not clear whether this might lead to any significant simplification of any of the proofs; the ability to use only single-step derivations may be of use in proving some of the results, but it is uncertain whether a definition of $F[\ ] \overset{t}{\underset{s}{\Rightarrow}} F'[\ ]$ using only s and t of length 0 or 1 would be an adequate "causing" property. All of the results presented in chapter 4 were researched before the emergence of the alternative definition of observation

equivalence; had this not been the case it is likely that $\approx_P$ would have been used in preference to $\approx$.

A second limitation to the application of the results presented in this thesis is the fact that the algorithm for the construction of bisimulations often involves lengthy checking, and it will be necessary to mechanise the algorithm before attempts are made to apply it to large systems. However, before any mechanisation is attempted it is desirable that a study be made of conditions sufficient to ensure the termination of the algorithm. It is not anticipated that this will be too difficult for the pure calculus, but it is likely that the admission of variables and parameterised behaviours would lead to additional complications. After an investigation of the termination of this algorithm has been made the process of mechanisation should not be unduly difficult. The program would have to be written in a language admitting recursively-defined types: it may be appropriate to define these using constructor and destructor functions in a language such as POP-2 [B+P] which combines properties of applicative and purely functional languages to some extent, or it might be found that a strongly-typed functional language such as ML is more suitable, possibly making use of the LCF system [GMW].

Another topic for further research is to investigate how the results and techniques presented in this thesis may be extended to the generalisations of CCS to synchronous and asynchronous calculi introduced by Milner in [Mil5]. It is not clear whether the definition of confluence could be extended easily to the synchronous calculus; the concepts involved appear to be less intuitive and properties of possible definitions would probably have to be investigated in some depth to inspire an appropriate approach. The concept of "causing" in the treatment of uniqueness criteria can probably be extended without too much difficulty, but it may be found that more work is required to establish that it satisfies all the desired properties, as the defintion of a derivation in the synchronous calculus involves a notion of

"time".

A further area for research is to attempt to relate the results proved in this thesis to the recent work of Hennessy and Plotkin [H+P], whose study of a term model for CCS to present a denotational semantics demanded the consideration of <u>least</u> fixed points of recursive behaviour definitions, under a partial ordering relating to "information content", which they obtained by introducing an operational preorder $\lesssim$ defined on behaviour expressions in a similar manner to $\simeq$, but taking divergence into consideration. Their research was performed at the same time as the criteria for unique fixed points of behaviour transformations presented in chapter 4 were being investigated; to relate the two studies it would be interesting to discover how the concepts of "causing" and the associated measure might be adapted to this preorder, and hence whether the conditions imposed on contexts in theorem 4.20 are sufficient to imply uniqueness of fixed points with respect to the equivalence induced by the Hennessy-Plotkin preorder.

It has been seen that many opportunities for future development have emerged from the work presented in this thesis; in addition to extending the applicabilty of the techniques introduced, further research aimed at putting the theory into a more general setting may be of interest. Indications have been made as to how this might be done regarding the relationship between the contracting transformations studied in chapter 4 and the concept of contraction mappings in metric spaces; attempts at such generalisation may possibly give some indication as to how the extensions to a wider class of transformations may best be performed. Other possibilities for future research include a detailed investigation into the relationship between confluence in CCS and the related concepts discussed in chapter 3: Huet's confluent reductions and the Church-Rosser properties of the λ-calculus.

In conclusion, it has become clear that, while the ideas and results introduced in this thesis seem to present a useful framework for the development of practical proof techniques for non-trivial systems of agents in CCS, there is considerable scope for further development in many directions.

ACKNOWLEDGEMENTS

I am indebted especially to my supervisor, Robin Milner, for many helpful suggestions and discussions on my research. Thanks are also due to George Milne, who introduced me to the field of concurrency theory, and to Matthew Hennessy, Gordon Plotkin and anyone else at the University of Edinburgh whose comments or questions may have influenced or helped to motivate my research.

Most of the work for this thesis was performed with the support of a research studentship from the UK Science and Engineering Research Council.

BIBLIOGRAPHY

[A+N] A .Arnold and M .Nivat
Metric Interpretations of Infinite Trees and Semantics of Non-deterministic Recursive Programs
in Theoretical Computer Science 11, 1980

[B+P] R. M. Burstall and R. Popplestone
POP-2 Reference Manual
in Machine Intelligence 2, Oliver + Boyd, 1968

[C+H] R. H. Campbell and A. N. Habermann
The Specification of Process Synchronisation by Path Expressions
in Lecture Notes in Computer Science 16, Springer-Verlag, 1974

[GLT] H. J. Genrich, K. Lautenbach and P. S. Thiagarajan
Elements of General Net Theory
in Lecture Notes in Computer Science 84, Springer-Verlag, 1980

[GMW] M. J. C. Gordon, A. J. R. G. Milner and C. P. Wadsworth
Edinburgh LCF
Lecture Notes in Computer Science 78, Springer-Verlag, 1979

[HLS] J. R. Hindley, B. Lercher and J. P. Seldin
Introduction to Combinatory Logic
London Mathematical Society Lecture Notes Series 7, Cambridge University Press, 1972

[Hoa] C. A. R. Hoare
Communicating Sequential Processes
in CACM 21.8, 1978

[Hue] G. Huet
Confluent Reductions: Abstract Properties and Applications to Term Rewriting Systems
in JACM 27.4, 1980

[H+M] M. C. B. Hennessy and A. J. R. G. Milner
On Observing Nondeterminism and Concurrency
in Lecture Notes in Computer Science 85, Springer-Verlag, 1980

[H+P] M. C. B. Hennessy and G. D. Plotkin
A Term Model for CCS
in Lecture Notes in Computer Science 88, Springer-Verlag, 1980

[K+C] R. B. Kolstad and R. H. Campbell
Path Pascal User Manual
in ACM Sigplan Notices 15.9, 1980

[Mil1] A. J. R. G. Milner
Processes: A Mathematical Model of Computing Agents
in Proc. Logic Colloqium 1973, N Holland, 1973

[Mil2] A. J. R. G. Milner
Synthesis of Communicating Behaviour
in Lecture Notes in Computer Science 64, Springer-Verlag, 1978

[Mil3] A. J. R. G. Milner
Flowgraphs and Flow Algebras
in JACM 26.4, 1979

[Mil4] A. J. R. G. Milner
A Calculus of Communicating Systems
Lecture Notes in Computer Science 92, Springer-Verlag, 1980

[Mil5] A. J. R. G. Milner
Calculi for Synchrony and Asynchrony
Internal Report CSR-104-82, Department of Computer Science, University of Edinburgh, 1982
(to appear in Theoretical Computer Science)

[M+M] G. J. Milne and A. J. R. G. Milner
Concurrent Processes and their Syntax
in JACM 26.2, 1979

[Par] D. J. Park
Concurrency and Automata on Infinite Sequences
in Lecture Notes in Computer Science 104, Springer-Verlag, 1981

[Pet] C. A. Petri
Concurrency
in Lecture Notes in Computer Science 84, Springer-Verlag, 1980

[Sut] W. A. Sutherland
Introduction to Metric and Topological Spaces
Clarendon Press, Oxford, 1975

[Tar] A. Tarski
A Lattice-theoretical Fixpoint Theorem and its Applications
in Pacific Journal of Mathematics 5, 1955